НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедри обчислювальної техніки

До захисту допущено:
Завідувач кафедри
Сергій СТІРЕНКО
«___»_____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»
на тему: «Спосіб перевірки цілісності відео файлів»

Виконав:
студент IV курсу, групи ІО-64
Кішка Марія Ігорівна _____

Керівник:
Асистент
Регіда Павло Геннадійович _____

Консультант з нормоконтролю:
Професор кафедри ОТ, д.т.н.
Сімоненко Валерій Павлович _____

Рецензент:
Посада, науковий ступінь, вчене звання,
Прізвище, ім’я, по батькові _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	62	
3	А4	ІАЛЦ.467200.003 ОА	Опис альбому	1	
4	А4	ІАЛЦ.467200.003 ТЗ	Технічне завдання	1	
5	А4	ІАЛЦ.467200.004 Д1	Додаток 1	1	
6	А4	ІАЛЦ.467200.005 Д2	Додаток 2	1	
7	А4	ІАЛЦ.467200.006 Д3	Додаток 3	1	
8	А4	ІАЛЦ.467200.007 Д4	Додаток 4	12	

				ІАЛЦ.467200.003		
	ПІБ	Підп.	Дата			
Розробн.	Кішка М. І.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Регіда П. Г.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІО-64	
Н/контр.	Сімоненко В. П.					
Зав.каф.	Стіренко С. Г.					

Пояснювальна записка
до дипломного проєкту
на тему: «Спосіб перевірки цілісності відео файлів»

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій СТІРЕНКО
«___»_____ 2020 р.

ЗАВДАННЯ
на дипломний проєкт студенту
Кішці Марії Ігорівні

1. Тема проєкту «Спосіб перевірки цілісності відео файлів», керівник проєкту асистент Регіда Павло Геннадійович, затверджені наказом по університету від «07» травня 2020 р. №1081-с.

2. Термін подання студентом проєкту: 15.06.2020.

3. Вихідні дані до проєкту: технічна документація. теоретичні та статистичні дані.

4. Зміст пояснювальної записки: аналіз та огляд існуючих методів перевірки цілісності, побудова та реалізація алгоритму перевірки на основі щілинної зйомки та програмна реалізація створеного методу.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): 29 рисунків, 3 блок-схеми.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В. П. професор кафедри ОТ, д.т.н.		

7. Дата видачі завдання: 09.11.2019.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	01.09.2020	
2	Вивчення та аналіз завдання	10.11.2019	
3	Розробка архітектури та загальної структури систем	17.01.2020	
4	Розробка структур окремих підсистем	07.02.2020	
5	Програмна реалізація системи	05.03.2020	
6	Оформлення пояснювальної записки	10.05.2020	
7	Захист програмного продукту	15.05.2020	
8	Передзахист	31.05.2020	
9	Захист	19.06.2020	

Студент Марія КІШКА

Керівник Павло РЕГІДА

АНОТАЦІЯ

В бакалаврській дипломній роботі було реалізовано та розроблено новий алгоритм перевірки цілісності відео файлів. Розглянуто найбільш розповсюджені причини пошкодження даних та головні методи їх виявлення. Алгоритм заснований на просторово-часовому представлені відео файлу. Такий метод дозволяє ефективно визначити наявність пошкоджень та дефектів на відео кадрах.

Створений алгоритм використовує метод щілинної зйомки для сканування файлів. Для дослідження відео файлів за допомогою щілинної зйомки в середовищі Borland Delphi 7 на мові програмування Object Pascal розроблено програмний засіб, що дозволяє власноруч отримувати фотографії, створені таким методом та проводити сканування. У роботі представлено схематичні ілюстрації та приклади зображень, створених з використанням ефекту щілинної зйомки. Програма може використовуватися різними користувачами для перевірки відео фрагментів на наявність пошкоджень.

ANNOTATION

In this work for a Bachelor's Degree the new algorithm for checking the integrity of video files was implemented and developed. The most common causes of data corruption and the main methods of their detection are considered. The algorithm is based on the space and time representation of the video file. This method allows to effectively determine the presence of damage and defects in video.

The created algorithm uses the slit-scanning method to scan files. The software for studying video files was developed in the Borland Delphi 7 environment using the Object Pascal programming language. Software that has been developed allows to manually obtain photos created by using this method and perform scans. The paper presents schematic illustrations and examples of images created using the slit-scan effect. The program can be used by various users to check video clips for damage.

ЗМІСТ

ВСТУП

У сучасному орієнтованому на цифрові дані світі, велику роль відіграє безпека інформації. Захист даних – це тільки одна частина процесу. Як тільки створені дані стануть потенційно корисними та конфіденційними даними постає питання, чи можна гарантувати їх оригінальність комусь ще. Актуальною темою сьогодення є пошук і впровадження нових можливостей зберігання і використання фото та відео інформації, що можуть бути забезпечені рівнем сучасних обчислюваних можливостей.

Багато сучасних та доступних методів сьогодення потребують від локальних системних адміністраторів чи команди безпеки впровадження елементів управління для захисту інформації. Проте організація такого процесу потребує окремих зусиль кожної структурної складової системи. Важливо зауважити, що навіть найбільш строгі міри безпеки даних не дозволяють вирішити питання оригінальності даних.

Звичайно, для контролю цілісності даних та перевірки даних на пошкодження використовується створення контрольних сум. Він може бути створений задля перевірки як і збережених так і переданих даних. Проте іноді неможливо отримати контрольну суму оригінального файлу, але існує велика необхідність перевірки достовірності даних файлу.

На сьогодні популярність використання відео даних, зйомка, збереження, редагування та поширення файлів через мережу Інтернет стрімку зростає. Стають популярними у всьому світі Інтернет-ресурси, що дозволяють будь-якому користувачу переглядати відео фрагменти та навіть завантажувати свої. Дуже модним стає створення відео роликів із конференціями, запис важливих заходів, бізнес зустрічей та поширення їх усім, хто їх потребує. Однак проблема цілісності відео файлів нікуди не зникає.

Саме тому задля покращення ефективності збереження цілісності відео файлів та ліквідації зайвого процесу перегляду відео фрагменту на перевірку його цілісності та відсутності помилок та артефактів було розроблено алгоритм

перевірки відео файлу. Розроблений алгоритм являє собою зручний інструмент сканування кожного окремого кадру з відео файлу на пошук помилок.

Алгоритм буз заснований на цікавому ефекті щілинної зйомки. Цифрова щілинна зйомка використовує набір від декількох сотень до декількох тисяч окремих кадрів для створення єдиного зображення, яке складається з елементів кожного із кадрів, і має істотний потенціал для фіксації великого обсягу інформації порівняно з традиційною серійною зйомкою. З іншої точки зору цифрова щілинна зйомка – це цікава, але малопоширена техніка, яка не вивчається у технічних ВНЗ, і тому на сьогоднішній день існує дефіцит інформаційних ресурсів, присвячених цій технології, та програмних засобів для її реалізації.

У роботі буде досліджено алгоритми, особливості та специфіку застосування даного методу при обробці відео, а також буде зроблено короткий опис аналогових засобів створення фотографій методом щілинної зйомки. У роботі представлено схематичні ілюстрації, що дають пояснення ефектам та методам сканування відео файлів.

РОЗДІЛ 1 ЦІЛІСНІСТЬ ТА ПОШКОДЖЕННЯ ФАЙЛІВ

1.1. Цілісність інформації

Точність та послідовність збережених даних, що вказується відсутністю будь-яких змін у даних між двома оновленнями запису даних називається цілісністю інформації. Цілісність даних накладається в базі даних на етапі її проектування за допомогою використання стандартних правил та процедур і підтримується за допомогою використання процедур перевірки помилок та їх усунення.

Поняття цілісності даних є підтримкою та забезпеченням точності і послідовності даних протягом усього його життєвого циклу і є одним з найважливіших аспектів в розробці, реалізації та використанні будь-якої системи, яка зберігає, обробляє або отримує дані [1]. Термін має широке охоплення і може мати істотно різні значення в залежності від конкретного контексту. Цілісність даних є протилежністю поняття пошкодження даних. Загальна мета будь-якої методики цілісності даних одна: забезпечити, щоб дані були записані точно за призначенням і після подальшого переміщення або запису, переконатися, що дані, не були змінені. Іншими словами цілісність даних спрямована на запобігання ненавмисного зміни інформації.

Будь-які непередбачені зміни в даних, як в результаті зберігання, пошуку або операції обробки, в тому числі зловмисно, або через несподіваний збій обладнання, через людської помилки є помилка цілісності даних. Залежно від даних, це може проявлятися у зміні в зображенні кольору одного пікселя, або навіть призвести до втрати безлічі цифрових файлів або критично важливих бізнес-бази даних.

1.2. Пошкодження даних

Пошкодження даних, в загальному, є псування або пошкодження комп'ютерних даних, викликаних людським, апаратним фактором та через помилки програмного забезпечення. Пошкодження даних не відбувається саме по собі. Практично завжди винні сили, які переміщують, контролюють і зберігають дані. Коли інформація передається в електронному вигляді, дані можуть пошкодитися [2]. У простій мережі інформація повинна переходити від жорсткого диска, звідти до шини, потім до пам'яті, потім до мережевої картки, потім до кабелю і назад, мільярди разів на день. Якщо будь-яке посилання в цьому ланцюжку несправне, інформація може бути втрачена або зіпсована. Найпоширенішими ознаками факту пошкодження даних на комп’ютері є:

- Відсутність або зміна положення файлів
- Помилка зчитування даних (наприклад недійсний формат файлу)
- Зміна назви файлу на невідомі символи (помилка кодування)
- Зміна дозволів та атрибутів файлу
- Несправна робота системи
- Повільна робота зовнішніх носіїв інформації

1.3. Найпоширеніші причини пошкодження даних та їх уникнення

Некоректне або аварійне завершення роботи. Однією з найбільш поширених причин пошкодження даних є неправильний виходом з програми. Це стосується вимикання або перезавантаження комп'ютера без використання належної процедури відключення. Це може бути як і навмисно зроблено, так і ненавмисно, при різкому відключенні електроенергії. Дослідження IBM показало, що для типового комп'ютера щомісяця виникає більше 120 проблем з живленням. Ці проблеми часто залишаються непоміченими, але можуть призвести до часткової втрати даних або збоїв на жорсткому диску. Вимкнення приладу з мережі при його роботі та відкритому програмному забезпеченні

може пошкодити дані. Це також стосується аварійного завершення роботи програми. Це спричиняє пошкодження файлів, наприклад коли файл на половині прогресу запису або переміщення цей процес різко зупиняється, створений файлу стає неповним та виникає пошкодження файлу. Правильне завершення роботи програми та коректне вимкнення приладу є основним вирішенням даної проблеми.

Фізичні дефекти. У цьому випадку фізичний дефект означає апаратну не функціональність, яка призводить до пошкоджених даних, що обробляються або зберігаються. Загальні причини несправного обладнання зазвичай полягають у механічному зносі та зносі компонента з часом. До більш серйозних можна віднести збій головки диска та наявність «битих» секторів, які можуть призвести до постійного пошкодження файлів, що записуються. Вчасна діагностика та перевірка справності апаратних складових є запорукою збереження цілісності даних.

Наявність шкідливого програмного забезпечення. Шкідливе програмне забезпечення відноситься до другої за частотою причини. Шкідливе програмне забезпечення, як правило, відноситься до будь-якої програми, призначеної для нанесення шкоди системі [3]. Такі небезпечні віруси можуть виконувати небажані операції, такі як видалення файлів, зміна даних, спричинення помилок запису та зчитування, знищення файлів документів або системних файлів, тощо. Деякі види шкідливого програмного засобу навмисно пошкоджують дані шляхом перезапису частини або всіх даних сміттям або випадковими даними. Проте існують програми відновлення даних для окремих випадків.

1.4. Перегляд існуючих способів перевірки цілісності файлів

Під час копіювання інформації або передачі її по мережі не гарантується її цілісність, що особливо актуально для великих обсягів інформації. Звичайно, можна порівняти розмір отриманого файлу з вихідним, але цього не достатньо, щоб стверджувати про ідентичність двох файлів. Тому були розроблені спеціальні алгоритми, що дозволяють вирішити цю задачу.

Найрозповсюдженим методом є перевірка контрольної суми даних. Контрольна сума, так само відома, як хеш сума, хеш-код або хеш файлу - це деяке значення, отримане за допомогою застосування певного алгоритму, що складаються з чисел, букв і знаків, що дозволяють упевнитися в цілісності і достовірності даних.

Контрольна сума - є унікальним ідентифікатором кінцевого набору даних (файлу), в якому містяться відомості про структуру даних, що дозволяють перевірити дані на цілісність або наявність помилок, відносно оригіналу, шляхом порівняння хеш сум до і після передачі даних, або під час зберігання. Алгоритмів хешування існує безліч, деякі з найбільш часто вживаними є: MD5, CRC32, SHA-1, SHA256, BTIH. Хеш-функція загалом призначена для згортки вхідного масиву будь-якого розміру в рядок бітів. Вона використовується наприклад для швидкого порівняння двох масивів на рівність: якщо у двох масивів хеші різні, то масиви гарантовано різні, а в разі рівності хеш - масиви швидше за все рівні. Однак найчастіше хеш-функції використовуються для перевірки унікальності пароля, файлу, рядка і тощо.

1.5. Види алгоритмів хешування

Алгоритм MD5. Алгоритм [4] був створений професором Рональдом Ріверстом в 1991 році на основі алгоритму MD4.

Для обробки MD5 отримує деякий рядок. Цей рядок перетворюється в послідовність з нулів і одиниць. Нехай q буде довжина отриманої

послідовності (64 біта, можливо, з незначущими нулями). До отриманої послідовності приписується 1. В результаті довжина послідовності збільшується на 1. Потім до послідовності приписуються нулі, поки довжина не стане по модулю 512 рівна 448 (довжина mod 512 = 448). Далі до послідовності дописують молодші 32 біта числа q, а потім - старші. Довжина послідовності стає кратною 512. Отриману послідовність назвемо S. Для підрахунку результату використовуються чотири подвійні слова (32 біта). Ці подвійні слова ініціалізуються наступними шістнадцятирічними значеннями, де першим йде наймолодший байт:

слово A: 01 23 45 67

слово B: 89 ab cd ef

слово C: fe dc ba 98

слово D: 76 54 32 10

Далі визначаються чотири допоміжні функції, які перетворюють вхідні 32-бітні слово у вихідні, також 32-бітні:

F(X,Y,Z) = X and Y or not(X) and Z

G(X,Y,Z) = X and Z or Y and (not(Z))

H(X,Y,Z) = X xor Y xor Z

I(X,Y,Z) = Y xor (X or not(Z))

На цьому етапі також реалізується підсилення алгоритму, яке складається з таблиці даних, що містить випадкові 64 числа, побудована наступним способом: $T[n]=int(2^{32}*|sin\ n|)$. Тобто в таблиці представлені по 32 біта після десяткової коми від значень функції sin, де аргумент n в радіанах.

Далі масив S розділяється на блоки по 512 бітів. Далі заносимо в блок даних елемент n з масиву 512-бітних блоків. Зберігаються значення A, B, C і D, що залишилися після операцій над попередніми блоками (або їх початкові значення, якщо блок перший). Кожний 512-бітний блок проходить 4 етапи обчислень по 16 раундів. Для цього блок представляється у вигляді масиву X з 16 слів по 32 біта (ABCD). Під час обчислення виконується послідовність операцій (функцій F, G, H, I) над масивом X. Всі раунди однотипні і мають

вигляд: [abcd k s i], який визначається як a = b + ((a + Fun (b, c, d) + X [k] + T [i]) \ <<< s) , де k - номер 32-бітного слова з поточного 512-бітного блоку, та <<< s - циклічний зсув вліво на s бітів отриманого 32-бітного аргументу. Число s задається окремо для кожного раунду. В кінці обчислення результати додаються та відбувається перехід до наступного блоку, якщо такий існує.

Після виконання цього алгоритму A, B, C, D - це результат (його довжина буде 128 біт). Часто можна бачити результат MD5 як послідовність з 32 символів 0..f. Це те ж саме, тільки результат записаний не в двійковій системі числення, а в шістнадцятирічній.

Наразі, існує безліч програм, що підбирають вихідне слово на основі хеша, що робить алгоритм MD5 ненадійним. Абсолютна більшість з них здійснює перебір по словнику, однак існують такі методи як RainbowCrack, що заснований на генеруванні безлічі хеш з набору символів, щоб по вийшла базі проводити пошук хеша. Також у MD5, як у будь-якої хеш-функції, існує таке поняття як колізії - це отримання однакових хеш для різних вихідних рядків.

Алгоритм SHA-1. Secure Hash Algorithm 1 - алгоритм криптографічного хешування. Для вхідних даних довільної довжини алгоритм генерує 160-бітове хеш-значення, зване також дайджестом повідомлення. Використовується в багатьох криптографічних додатках і протоколах. Принципи, покладені в основу SHA-1, аналогічні тим, які використовувалися Рональдом Ривестом при проектуванні MD4.

SHA-1 реалізує хеш-функцію, побудовану на ідеї функції стиснення. Входами функції стиснення є блок повідомлення довжиною 512 біт і вихід попереднього блоку повідомлення [5]. Вихід є значення всіх хеш-блоків до цього моменту. Іншими словами хеш-блок M_i дорівнює $h_i = f(M_i, h_{i-1})$. Хеш-значенням всього повідомлення є вихід останнього блоку [5].

Оригінал тексту (даних) розбивається на блоки по 512 біт в кожному. Останній блок доповнюється до довжини, кратної 512 біт. Спочатку додається 1 (біт), а потім нулі, щоб довжина блоку стала рівною 512 - 64 = 448 біт. В

останні 64 біта записується довжина вихідного повідомлення в бітах. Якщо останній блок має довжину понад 448, але менше 512 біт, то додавання виконується в такий спосіб: спочатку додається 1 (біт), потім нулі до кінця 512-бітного блоку; після цього створюється ще один 512-бітний блок, який заповнюється до 448 біту нулями, а в останні 64 біта записується довжина вихідного повідомлення в бітах. Доповнення останнього блоку здійснюється завжди, навіть якщо повідомлення вже має потрібну довжину.

Далі відбувається ініціалізація п'яти 32-бітових змінних A, B, C, D, E.

A ← 67452301_{16}

B ← $EFCDAB89_{16}$

C ← $98BADCFE_{16}$

D ← 10325476_{16}

E ← $C3D2e1F0_{16}$

Потім виконується обробка кожного блоку. Для цього значення змінних A, B, C, D, E копіюються в змінні a, b, c, d, e і далі для t від 1 до 80 виконується перетворення значень даних змінних за схемою, зображеної на рис. 1.1.

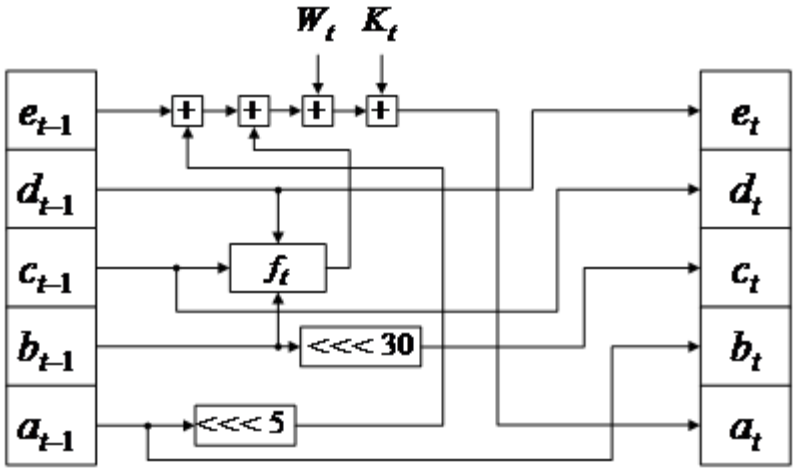


Рис. 1.1. Схема перетворення даних

Кожна з 80 ітерацій може бути записана наступним чином:

$TMP \leftarrow (a \lll 5) + f_t(b,c,d) + e + W_t + K_t;$

$e \leftarrow d;$

$d \leftarrow c;$

$c \leftarrow (b \lll 30)$;

$b \leftarrow a$;

$a \leftarrow TMP$;

де «+» - операція додавання по модулю 2^{32}, $f_t(b, c, d)$ - нелінійна функція, яка має наступний вигляд:

$$f_t(X,Y,Z) = \begin{cases} (X\&Y) \vee (!\,X\ \&\ Z), \\ X \oplus Y \oplus Z, \\ (X\ \&Y)|(X\ \&\ Z)|(Y\ \&\ Z), \\ X\ \oplus Y \oplus\ Z, \end{cases} \quad (1)$$

де «&» - побітова операція «І», «|» - побітова операція «АБО», «!» - інвертор, «⊕» - операція побітового додавання за модулем. Параметр K_t приймає чотири різних значення в залежності від номера поточної ітерації:

Kt = 5A82799916,

Kt = 6ED9EBA116,

Kt = 8F1BBCDC16,

Kt = CA62C1D616.

«⋘» - операція циклічного зсуву на 30 або 5 біт вліво, Wt - одне з шістнадцяти 32-бітних слів 512-бітного блоку даних при яких значення, яке визначається відповідно до наступним виразом при t від 1 до 16.

Значення змінних a, b, c, d, e незалежно один від одного складаються по модулю 2^{32} зі значеннями змінних A, B, C, D, E, в які потім і поміщаються отримані результати. Після обробки останнього блоку тексту значення хеш-образу формується як ABCDE.

З 2011 по 2015 рік основним алгоритмом хешування був SHA-1. Зростаюча кількість досліджень, що показують слабкі сторони SHA-1, викликала переоцінку. 2017 року фахівці компанії Google, за підтримки вчених з нідерландського Центру математики та інформатики, які працювали і над доповіддю про небезпечність використання даного алгоритму, представили на суд публіки звіт про першу вдало виконану колізійну атаку на SHA-1, що отримала ім'я SHAttered . Як доказ успіху атаки фахівці опублікували два PDF-

файлу з однаковим SHA-1 хешем. Через це більшість провідних компаній світу відмовились від використання SHA-1, тому на зміну йому прийшов алгоритм SHA-2.

SHA-1 і SHA-2 - це дві різні версії алгоритму Secure Hash Algorithm. Вони відрізняються як і за побудовою так і в бітовій довжині суми. SHA-1 - 160-бітний хеш. SHA-2 - це фактично «сімейство» хешів і має різну довжину, найпопулярніша - 256-бітна. Хеш-функції сімейства SHA-2 побудовані на основі структури Меркле - Дамгора.

Короткий опис: Оригінал тексту після доповнення розбивається на блоки, кожен блок - на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64 або 80 ітераціями (раундами). На кожній ітерації 2 слова перетворюються, а функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума і є значенням хеш-функції. Проте, ініціалізація внутрішнього стану проводиться результатом обробки попереднього блоку. Тому незалежно обробляти блоки і складати результати неможливо.

1.6. Огляд програмних засобів перевірки цілісності файлів

MD5 File Checker. Md5Checker - це безкоштовний, швидкий, легкий і простий у використанні інструмент для управління, обчислення та перевірки контрольної суми MD5 з декількох файлів / папок [6]. Розмір додатку становить близько 300 Кб і не потребує встановлення (портативний). На рис. 1.2 зображено інтерфейс даного програмного засобу.

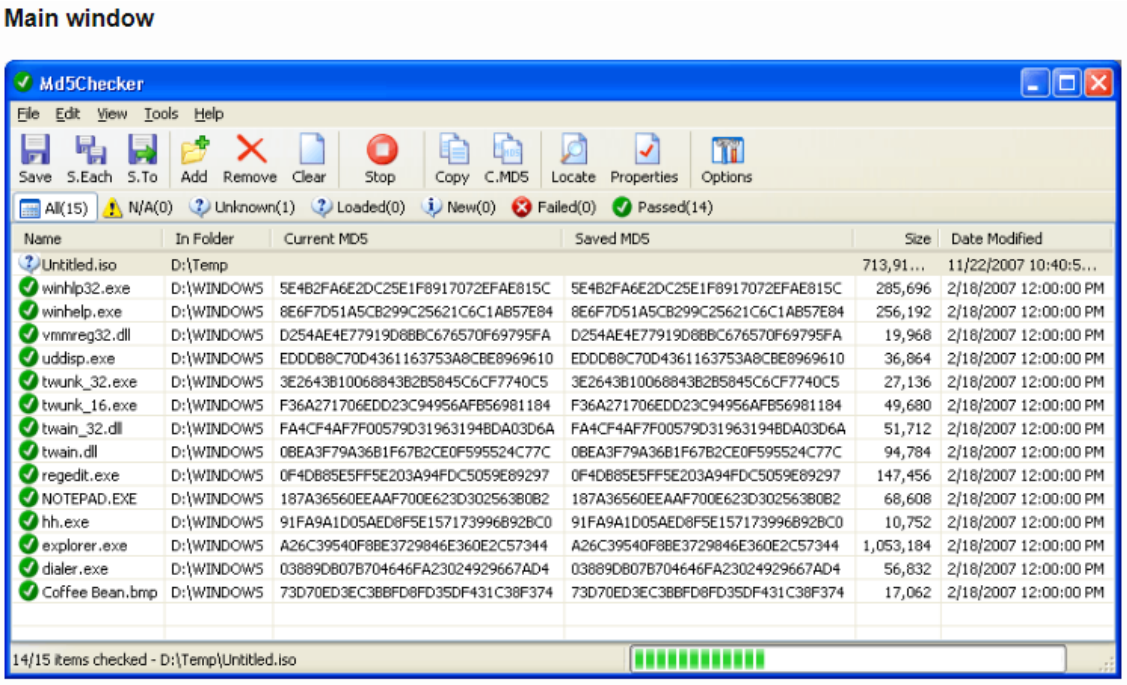


Рис. 1.2. Скріншот головного вікна програми

Основні функції даного програмного засобу:

- Перевірка цілісності завантажених файлів: За допомогою Md5Checker користувач може обчислити контрольну суму MD5 завантажених файлів і порівняти їх із передбаченими при завантаженні через HTTP, FTP, P2P тощо.
- Для виявлення невідомих вірусів: антивірусні програми, які працюють з певною базою даних вірусів, рідко можуть вчасно виявити всі нові (тобто не з бази даних) віруси. Md5Checker перевіряє, чи файли є оригінальними. За допомогою цього користувач може виявити будь-яку зміну файлу, включаючи вірус.
- Щоб переконатися, що інсталяційні файли надійно захищені: система буде повторно заражатися вірусами під час установки програмного забезпечення, якщо інсталяційний файл був заражений. Щоб уникнути цього, функціонал додатку дозволяє обчислити та зберегти контрольну суму MD5 відразу після завантажених / скопійованих інсталяційних файлів.

- Перевірка стану безпеки системи: система була заражена, якщо контрольна сума MD5 одного виконуваного файлу була змінена без жодних дій.
- Щоб з'ясувати джерело вірусу: програма може вказувати на те, що один виконуваний файл є джерелом вірусу, якщо контрольна сума MD5 інших кількох виконуваних файлів несподівано змінилася після виконання цього файлу.
- Для обчислення контрольної суми MD5 для публікації: Розповсюджувачі файлів та автори програмного забезпечення можуть використовувати Md5Checker для обчислення контрольної суми MD5 своїх файлів та публікації їх на веб-сайті.

EF CheckSum Manager. EF CheckSum Manager - це програма, призначена для перевірки цілісності файлів у стандартних форматах SFV, MD5 та SHAx [7]. Вони можуть перевірити наявні контрольні суми або створити нову контрольну суму для важливих даних користувача. Менеджер EF CheckSum простий у використанні та швидкий у виконанні, підтримує рекурсивну обробку файлових структур та може опрацьовувати усі накопичувачі. У додатку можна створити один файл контрольної суми для всіх файлів, по одному на папку, або створити для кожного файлу окремі контрольні суми.

Додаток має стандартний набір функцій (див. рис. 1.3) – це створення контрольних сум для файлів перед їх архівацією, передачею через інтернет, тощо.

Мінімальні вимоги для встановлення: Комп'ютер Pentium, Microsoft Windows 32/64 біт, 2 Мб вільного місця на жорсткому диску.

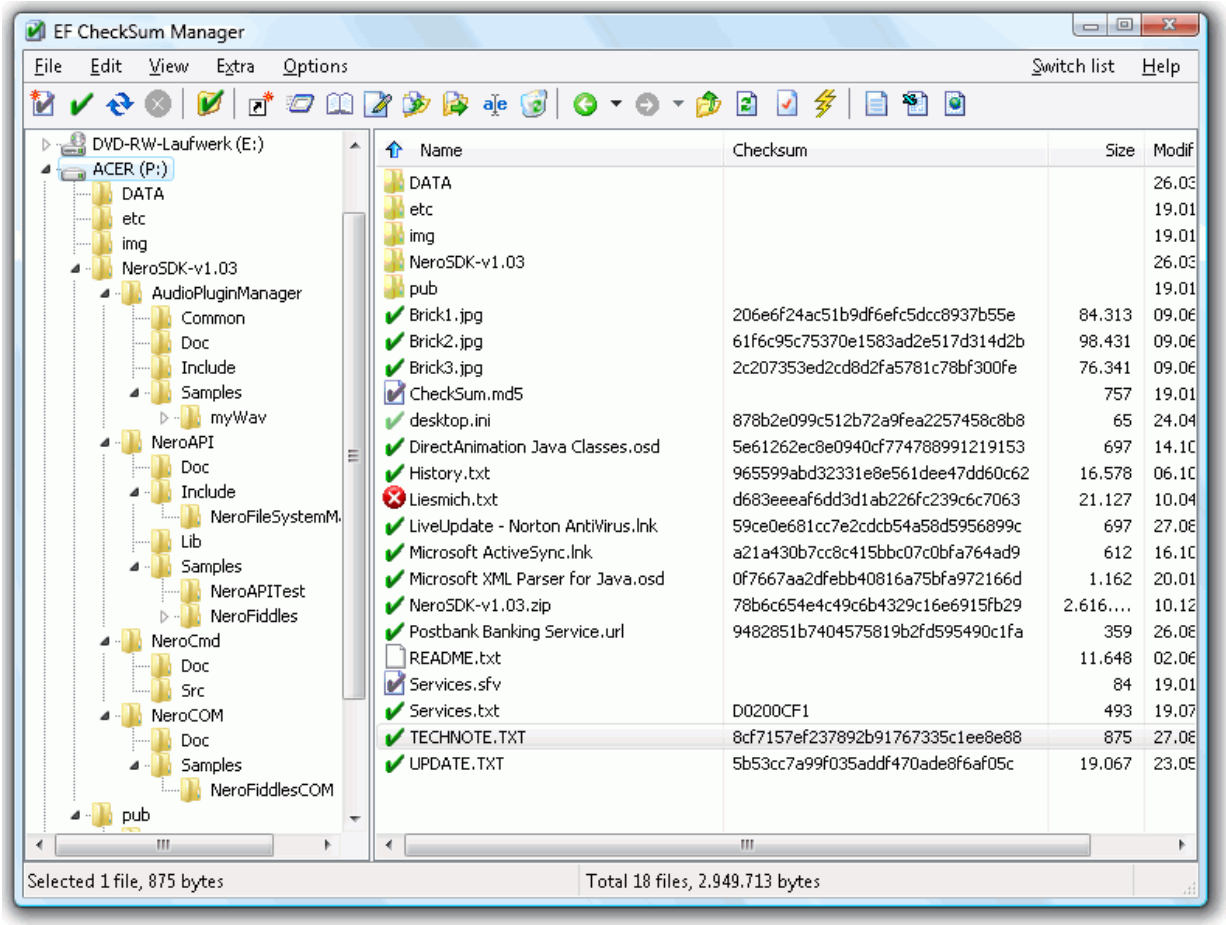


Рис. 1.3. Скріншот головного вікна програми EF CheckSum Manager

WinMD5Free. Програма перевіряє цілісність файлів, розраховуючи значення контрольних сум MD5 для файлів [8]. Програмний додаток не залишає слідів у реєстрі комп'ютера, його можна з легкістю видалити. Є можливість звернутися до довідкового матеріалу, щаб налаштувати деякі параметри вручну.

Особливості:

- Підтримує майже всі платформи Windows, включаючи Microsoft Windows 95, 98, 2000, Me, XP, 2003, Vista, Windows 7, 8 та Windows 10.
- Підтримуються великі файли розміром більше 4 Гб.
- Низьке використання ресурсів. Він використовує менше 5 Мб оперативної пам'яті.
- Не потрібно встановлювати .NET час виконання. Це окремий файл EXE, і запуск швидкий.

- Підтримується „Перетягування”. Ви можете або вибрати файл, або перетягнути файл у вікно програми, щоб отримати хеш-значення MD5.
- Підтримує перевірку початкового значення MD5 та поточного значення MD5.
- Невеликий розмір, ефективний і крихітний інструмент для захисту даних. Також безкоштовний додаток.

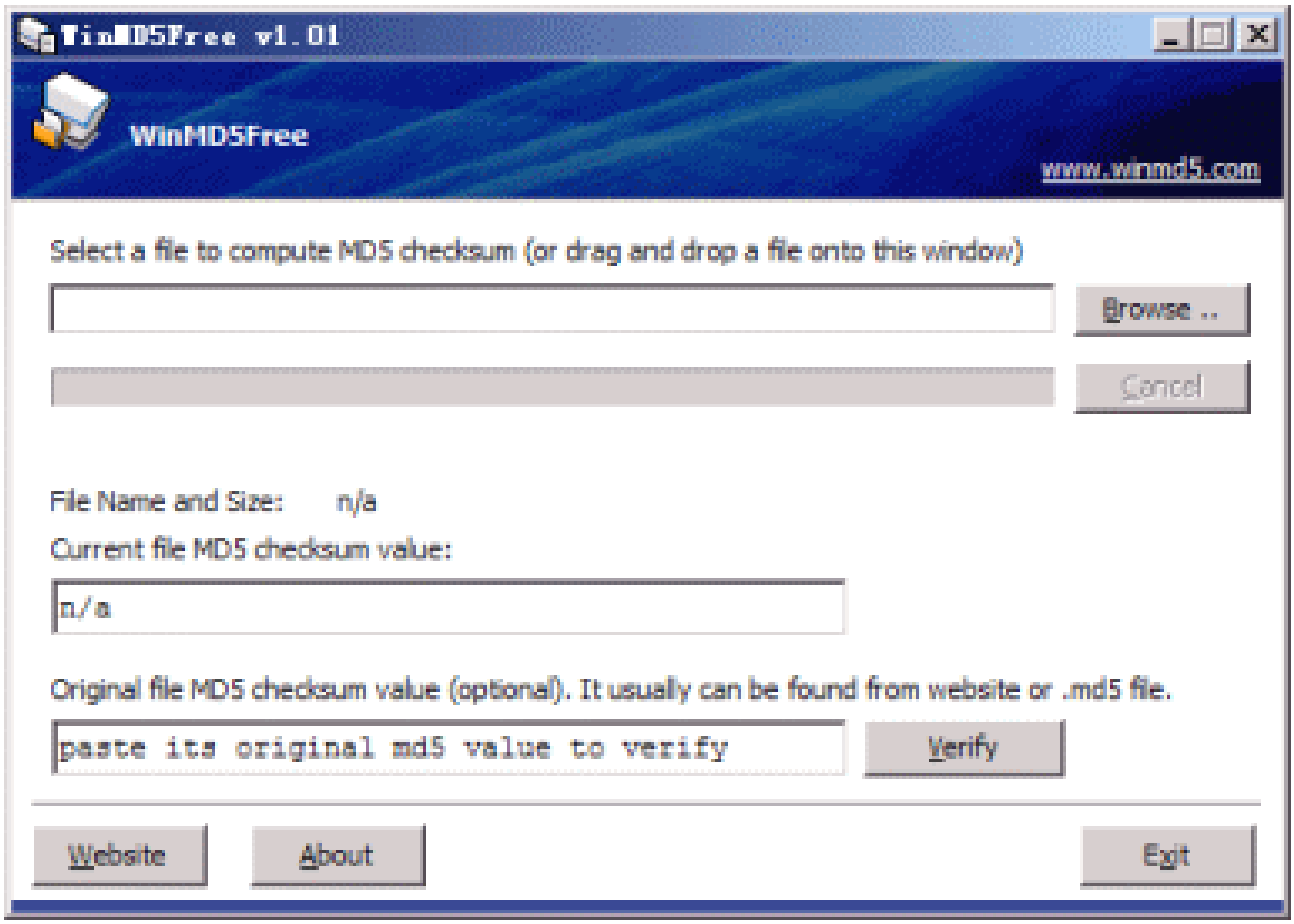


Рис. 1.4. Інтерфейс програми WinMD5Free

CimTrak. CimTrak Integrity Suite [9]– це повноцінна система захисту, перевірки та забезпечення цілісності великих баз даних, що призначена для великих IT-компаній.

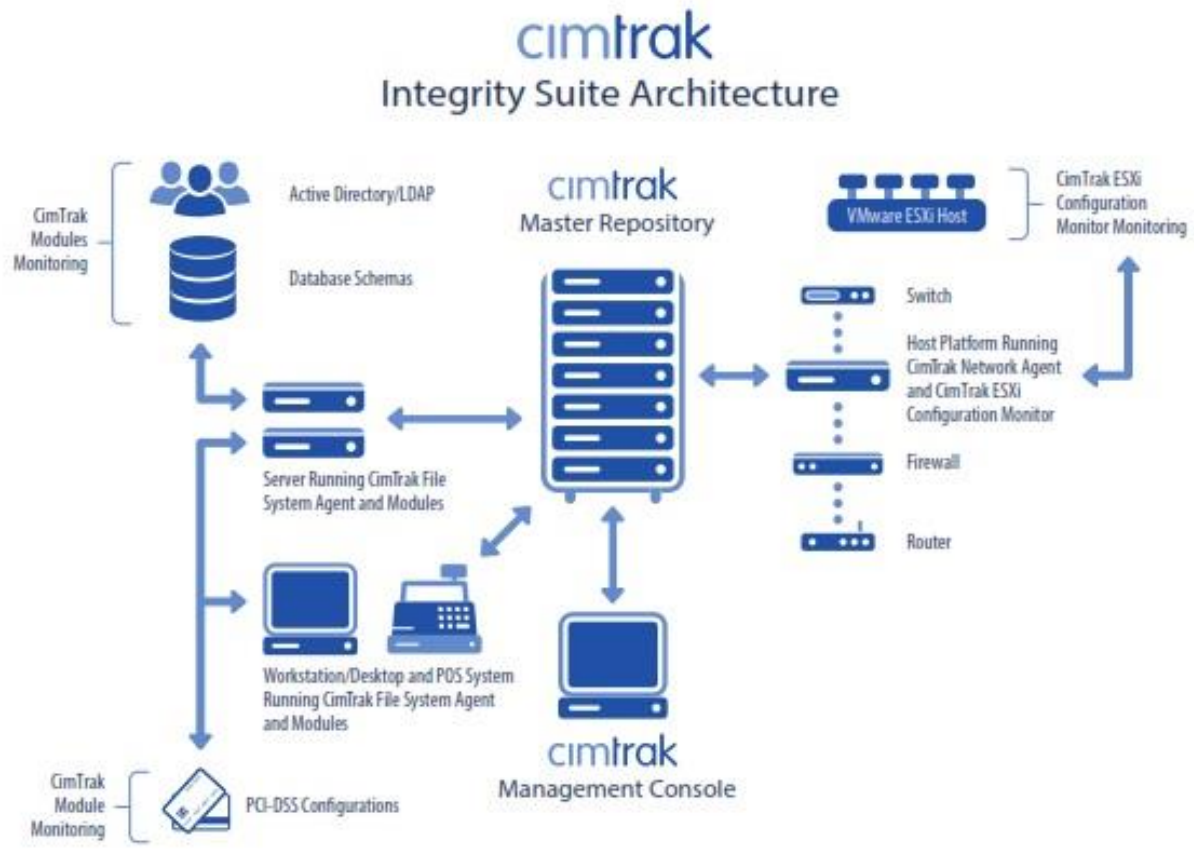


Рис. 1.5. Схема роботи системи CimTrak

Основні особливості:

- Виявлення усіх змін у IT-середовищі з охопленням серверів, мережевих пристроїв, критичних робочих станцій, систем продажу, тощо. CimTrak забезпечує просте налаштування та управлінням системи, яка функціонує, як єдиний пункт збору та звітування про зміни, які можуть вплинути на операції, безпеку та відповідність.
- Отримання миттєвих повідомлень про зміни у системі. CimTrak дає глибоку ситуаційну обізнаність про те, що саме відбувається у вашому IT-середовищі. Це дає можливість миттєво усвідомлювати зміни та постійно знати про стан вашої критичної IT-інфраструктури.
- За необхідності можна дозволити CimTrak автоматично відреагувати та виправити виявлену проблему. Крім того, CimTrak надає можливість вживати миттєві, автоматичні дії для перенаправлення або запобігання змін повністю.

- Можливість надання звітування: документацію щодо змін у вашому агентстві чи на підприємстві. CimTrak надає повний масив звітів, як про зміни у вашому IT-середовищі, так і про дії, вжиті в CimTrak. CimTrak також легко експортує зібрану інформацію про зміни до різних інструментів звітування та оповіщення, присутніх на багатьох підприємствах та урядових установах, включаючи інформацію про безпеку та керівників подій (SIEM).

CimTrak не є безкоштовним рішенням для підприємств, проте має безкоштовну пробну версію. Розрахунок ціни відбувається окремо для кожної компанії, залежно від її розміру та інших факторів.

System File Checker. Це проста у використанні командна утиліта для ОС Windows, яка перевіряє наявність пошкоджених системних файлів та замінює їх на хорошу копію файлу. Цей вбудований інструмент для Windows використовується, коли є підозра на пошкодження системних файлів або якщо з системою щось не так [10].

Сканування запускається командою sfc/scannow у командній строці Windows. Коли сканування завершено, є чотири варіанта повідомлень:

Повідомлення 1: Захист ресурсів Windows не виявив порушень цілісності. Це означає, що у системі немає пошкоджених чи відсутніх системних файлів.

Повідомлення 2: Захист ресурсів Windows не зміг виконати запитувану операцію. Вищезгадану проблему можна вирішити, запустивши сканування SFC у безпечному режимі.

Повідомлення 3: Захист ресурсів Windows виявив пошкоджені файли та успішно їх відновив. Це означає що пошкоджені файли все ж таки були знайдені та вдало відновлені.

Повідомлення 4: Захист ресурсів Windows знайшов пошкоджені файли, але не зміг виправити деякі з них. Це означає, що потрібно вручну замінити пошкоджені файли.

Висновки до розділу 1

Важливим поняттям у інформаційному просторі є цілісність даних. Цілісність інформації визначається в точній та послідовній передачі інформації протягом усього його життєвого циклу. Це один із важливих факторів у розробці, створення та передачі інформації у будь-якій системі.

Протилежним поняттям є пошкодження даних. В результаті зберігання, пошуку або операцій обробки може виникнути збій або помилка, що призведе до пошкодження даних. Пошкодження даних відбувається не без зовнішніх факторів. Перш за все такі помилки можуть виникати через людську необачливість або через апаратний збій обладнання. Більшість з таких причин виникнення порушення цілісності даних були розглянуть у розділі.

Найрозповсюдженим методом перевірки цілісності даних є використання алгоритмів хешування задля перевірки контрольних сум. Контрольна сума – це унікальний набір символів чи літер, що ідентифікує файл та містить усі відомості про нього. Популярні методи хешування, тобто створення контрольних сум, такі, як MD5, SHA-1, SHA-2 були детально описані у розділі.

Також було розглянута деякі програмні засоби, що дозволяють перевірити цілісність інформації. Було виявлено їх переваги та недоліки та проаналізовано їх працездатність.

РОЗДІЛ 2 МЕТОДИ ПЕРЕВІРКИ ЦІЛІСНОСТІ ВІДЕО ФАЙЛІВ

2.1. Декодування файлів

Немає простого способу дізнатися файл пошкоджений чи ні. Принаймні, якщо ви не знаєте контрольної суми (наприклад, хеш MD5 / SHA-1 або принаймні значення CRC) "правильного" вихідного файлу.

Більшість форматів аудіо та відео контейнерів просто не містять контрольних сум, тому немає ніяких способів перевірити їх цілісність.

Все, що можна зробити - це спробувати розшифрувати повний файл і побачити, чи розшифровує декодер несподівані помилки / попередження.

Дешифратор або декодер — логічний пристрій, який перетворює код числа, що поступило на вхід, в сигнал на одному з його виходів. Вихідними функціями дешифратора є різноманітні конституенти одиниці [11].

Якщо число представлено у вигляді n двійкових розрядів, то дешифратор повинен мати 2^n виходів. Дешифратор довільної складності може бути складено з трьох базових логічних елементів: кон'юнкції, диз'юнкції та заперечення.

За принципом дії розрізняють такі види дешифраторів [12]:

- Послідовні,
- Паралельні,
- Паралельно-послідовні.

Розрізняють дешифратори першого та другого роду:

- Дешифратори першого роду реалізують систему функцій, кожна з яких приймає одиничне значення при відповідному одиничному значенні вхідного слова.
- Дешифратори другого роду реалізують систему функцій, кожна з яких приймає одиничне значення при визначених діапазонах вхідного слова.

За способом побудови розрізняють:

- Лінійні дешифратори

У такому дешифраторі *n* змінних, представляють сукупність не зв'язаних між собою 2^n систем збігу на *n* входів, кожна з яких реалізує відповідну конституенту одиниці.

- Пірамідальні дешифратори

Будуються за принципом послідовних каскадів: на першому каскаді реалізуються конституенти одиниці для 2 змінних, на n-1 реалізуються конституенти одиниці для n змінних, при цьому, на вході отримується вихід з попереднього каскаду.

2.2. Огляд існуючих форматів відео та відео кодеків

При створені відео дуже велику роль грають не тільки параметри розміру, роздільної здатності чи відношення сторін. Потрібно також брати до у ваги формат відео файлу з урахуванням стиснення (компресії), яка властива різним форматам файлів.

Медіаконтейнер або мультимедійний контейнер – формат файлу або потоковий формат, специфікації якого визначають тільки спосіб представлення даних (а не алгоритм кодування) в межах одного файлу. Він визначає розмір та структуру файлів, що представляються. Медіаконтейнер фактично є метаформатом, так як він зберігає дані і інформацію про те, як дані будуть зберігатися всередині файлу [13]. Як наслідок, програма, яка здатна коректно ідентифікувати і відкрити файл (прочитати потік), записаний в якому-небудь форматі, внаслідок може бути не здатна декодувати фактичні дані, закодовані в медіа контейнері.

В теорії формат контейнер здатний зберігати будь-який тип даних, проте в реальності для окремих типів файлів існують окремо визначені групи контейнерів. Ці групи налаштовані для специфічних потреб і інформації, котра буди зберігатися в них. Медіа контейнери є типовим прикладом такої групи файлових контейнерів, котрі створені для зберігання інформації, медіа інформації, яка вдало поділяється на картинки, відео і аудіо.

У випадку фільмів медіа контейнер повинен не тільки зберігати відео та аудіо потоки, а також повинен містити мітки для синхронізації при відтворенні відео. У медіа контейнері може зберігатися декілька потоків одного типу, як приклад фільм, тобто відео потік) з декількома доріжками, тобто аудіо потоками, та субтитрами, тобто потоком тексту.

2.3. Поняття відео кодеку

Контейнер файлу використовується для ідентифікації та чередування різних типів даних. Також деякі формати контейнерів можуть зберігати в собі різні типи звукових даних, закодовані деяким кодеком.

Кодек – пристрій чи програма, яка здатна виконувати перевтілення потоків даних чи сигналів. Для зберігання, передачі чи шифрування потоку даних чи сигналу його кодують за допомогою кодека, а для перегляду чи редагування декодують. Кодеки дуже часто використовують при програмній обробці відео та звуку. В кодеках можуть використовуватися два види стиснення даних. Більшість таких відео кодеків надають перевагу стиснення з витратами, адже це значно змінює об’єм даних, тобто зменшує його, для зберігання або надсилання файлів. Проте такий вид стиснення призводить до зниження якості звуку чи картинки при відтворенні фільму. У випадку коли не рекомендована втрата даних, то кодеки змінюються та вже виконують стиснення без витрат. Це дуже корисно якщо потрібно надалі редагувати відео файл, адже для редагування потрібна найякісніша роздільна здатність та якість відео фрагменту [8].

Ключовим поняттям для вибору формату відео файлу є поняття стиснення. У сирих, тобто нестиснених відео файлів, як правило, занадто великий об’єм – відеоінформація досить складна і потребує багато місця для зберігання. Замість того, щоб змушувати користувачів працювати з величезними файлами, відео індустрія розробила концепцію стиснення файлів, призначену для зменшення їх розміру.

Кожний окремий метод стиснення сформований у вигляді кодеку. Чим менші по об’єму файли видає кодек, том більш ефективним він виявляється. Деякі кодеки також надають можливість вибору рівня стиснення. Як правило, вона виражається в швидкості відтворення відео файлу (найчастіше вимірюється в кілобайтах в секунду). Чим більше швидкість, тим краще якість файлу, проте від цього збільшується його розмір. Вибір формату файлу та кодеку – це компроміс між розміром файлу та якістю відтворення відео та аудіо.

Вибір кодеку важливий, адже різні формати файлів використовують різні форми стиснення так, що виникає величезна кількість сполучень для вибору. Формат файлу або медіа контейнер – це конкретний спосіб кодування цифрової інформації. Існує безліч різних файлових форматів для різного використання. Наприклад, формат PDF від Adobe використовується для створення потрібних для друку документів, а JPEG - для зберігання цифрових фотографій. У кожного файлового формату є свої унікальні характеристики, і, як правило, він несумісний з іншими, навіть аналогічними форматами.

2.4. Огляд і порівняння існуючих форматів відео файлів

Для зберігання відеоінформації використовується декілька різних файлових форматів. У кожного свої переваги і обмеження, так що потрібно ознайомитися з ними до того, як почати створювати відео. Загалом медіа контейнери є прикладом файлових контейнерів. Вони застосовуються для зберігання медіа інформації, до якої відноситься аудіо, відео та зображення. Медіа контейнер – це формат файлу, специфікації якого стосуються лише методу збереження даних в межах одного файлу.

Більш складні медіа контейнери можуть підтримувати множинні аудіо- і відео потоки, текстові субтитри, інформацію по розділах, метадані. У деяких випадках заголовок файлу, більшість метаданих і синхронізаційні дані зазвичай визначаються самим контейнером. Наприклад, є контейнери, створені

для відео з низькою роздільною здатність і є контейнери, оптимізовані для високоякісних файлів, що містять багато потоків високої якості.

Частини структури контейнеру відео файлу мають різні імена. У RIFF і PNG їх часто називають шматками, в MPEG-TS їх називають пакетами, а в JPEG вони називаються «сегменти». Основний контент даних складових частин називається даними або «корисним навантаженням». У більшості з них кожна складова частина має свій заголовок, в той час як медіа контейнер TIFF натомість зберігає зміщення, що призводить до складності при збереженні інформації. Модульні складові частини полегшують відновлення інших складових частин у разі пошкодження файлу або при «випаданні» кадрів. Медіа контейнери для зберігання різних даних представлені у таблицях 2.1 – 2.3.

Таблиця 2.1

Формат файлу	Розширення	Характеристики
FITS	.fits	Цифровий формат файлів, який використовується в науці для зберігання, передачі та обробки зображень і їх метаданих (електронних таблиць). Найчастіше FITS використовується в астрономії.
TIFF	.tif або .tiff	Графічний формат файлів, розроблений як один із основних універсальних форматів інсценізації зображень високої якості, які використовуються у видавничій галузі.

Таблиця 2.2 Медіа контейнери для зберігання аудіо даних

Формат файлу	Розширення	Характеристики
AIFF	.aif, .aiff або .aifc	Спеціальний формат файлів (аудіо), який потрібен для відтворення та зберігання даних звуку на носіях інформації та інших радіопристроях.
WAV	.wav	Формат файлу-контейнера призначений для зберігання запису оцифрованого аудіо потоку, підвид RIFF. WAV контейнер зазвичай використовується для зберігання короткого звуку в імпульсно-кодовій модуляції.
XMF	.xmf	Сімейство пов'язаних музичних форматів файлів. Створені для того, щоб включати в себе різні типи мультимедійних файлів, таких як файли даних MIDI (.MID), файли інструментів (.DLS) і аудіо хвиль (.WAV).

Більшість медіа контейнерів пристосовані для зберігання всіх або майже всіх типів медіа інформації. Найпопулярніші з них представлені у таблиці 2.3. Деякі з них також можуть зберігати в собі додаткову інформацію.

Таблиця 2.3

Формат файлу	Розширення	Характеристики
3GP	.3gpp, .3gpp2	Файловий формат для мультимедіа контейнерів, загальноприйнятих у використанні в багатьох телефонах. Заснований на стандартному форматі медіа файлів ISO.

Продовження Таблиці 2.3

Формат файлу	Розширення	Характеристики
H.264	.mpg, mp4	Формат стиснення, що використовується у багатьох файлах MPEG-4, 3GP та QuickTime. Більш ефективний, ніж звичайний кодек MPEG-4, проте записує високоякісне відео у відносно малі по розміру файли.
Matroska	.mkv	Відкритий проєкт, тобто безкоштовна для персонального використання. Її контейнер має змогу зберігати багато аудіо потоків, відео потоків і субтитрів.
MPEG-1	.mpg, .mpeg	Початковий формат MPEG. Надає відео низької якості, схоже на запис на відео касетах. У теперішніх реаліях використовується мало.
MPEG-2	.mpg, .mpeg	Наступна версія формату MPEG, що відтворює аудіо-відео кращої якості в порівнянні з MPEG-1. Використовується у телебаченні та деяких супутникових сервісах.
MPEG-4	.mpg, mp4	Остання версія формату MPEG, що оптимізована як і для високої роздільної здатності, так і для інтернет-відео. Використовується у більшості нових камкодерів. Може використовувати кодеки H.264 і DivX.

Продовження таблиці 2.3

Формат файлу	Розширення	Характеристики
MPEG-PS	.mpg, .mpeg	PS – це «програмний потік». Формат потокового відео, що використовується для потокового відтворення файлів MPEG-1 та MPEG-2.
QuickTime	.mov, .qt	Закритий аудіо та відео формат для Apple. Працює як і на MacOS, так і на ОС Windows.
RealVideo	.rm, .rt	Формат медіафайлів, що використовується у RealPlayer. Формат може зберігати як і відео так і аудіофайли. Цей формат з високою ступінню стиснення, який дає на виході відео більш низької якості у порівнянні з конкуруючими форматами.
WebM	.webm	Один з найтипових та новіших форматів відео, розроблений спеціально для зберігання матеріалів, не потребуючих лицензійних відрахувань. Розроблений для використання з HTML5-відео. Підтримується більшістю Інтернет браузерів.
Windows Media Video (WMV)	.wmv	Закритий файловий формат цифрового відео Microsoft, призначений для відтворення у Windows Media Player.

Продовження таблиці 2.3

Формат файлу	Розширення	Характеристики
Audio Video Interleave (AVI)	.avi	Формат контейнеру, що здатний зберігати дані, що стиснені різними кодеками.
DivX	.divx	Кодек з високою якістю зображення і високим ступенем стиснення. Є підмножиною формату MPEG-4, що підтримує роздільну здатність до 1080p. Популярний в мережі Інтернет.
Digital Video (DV)	.dv	Формат з витратами, що використовується у багатьох звичайних відео камерах. Відео з таким форматом зазвичай поміщаються в контейнери, такі як AVI або QuickTime.
Flash Video	.flv	Популярний формат мультимедіа файлів від Adobe. Використовується для відтворення відеороликів на сайті YouTube.

2.5. Декодування відео файлів для перевірки цілісності.

Для того, щоб перевірити відео файл на пошкодження, проте немає можливості визначити контрольну суму оригінального файлу, можна скористатися декодерами. За допомогою них можна отримати дані, що допоможуть виявити помилки. Це доволі складна операція, проте існують спеціальні фреймворки, які дозволяють це зробити. Декодери, в загальному,

можуть виявити помилки при дешифруванні відео файлу. Таким чином відео файл перевіряється на наявність пошкодження.

FFmpeg. Найрозповсюдженим додатком для дешифрування відео файлів є FFmpeg. FFmpeg - це провідний мультимедійний фреймворк, здатний розшифровувати, кодувати, декодувати, переглядати, фільтрувати та відтворювати майже все, що створили люди та машини. Він підтримує усі старовинні формати. Незалежно від того, чи були вони розроблені певним комітетом зі стандартів, громадою чи корпорацією. Він також дуже портативний: FFmpeg компілює, запускає та передає тестувальну інфраструктуру FATE через Linux, Mac OS X, Microsoft Windows, BSD, Solaris, тощо, у різних середовищах побудови, архітектури машин та конфігурацій.

Він містить наступні утілити: libavcodec, libavutil, libavformat, libavfilter, libavdevice, libswscale та libswresample, які можуть використовуватися програмами. А також ffmpeg, ffplay та ffprobe, які можуть бути використані кінцевими користувачами для перекодування та відтворення.

Проект FFmpeg намагається забезпечити найкраще технічно можливе рішення як для розробників програм, так і для кінцевих користувачів. Це один із доступних варіантів безкоштовного програмного забезпечення. Наразі проект знаходиться в стані розробки.

Для того щоб розпочати перевірку да дешифрування потрібно запустити наступний код на виконання. В ньому першочергово запускається утиліта FFmpeg, вказуються параметри, які обов’язково потрібні для процесу дешифрування, вказується файл, шлях до файлу, що перевіряється:

```
ffmpeg.exe -loglevel -f yuv4mpegpipe -i c: \ foo \ file_to_test.mkv -> NUL
```

Звичайно, цей код може виявити не всі пошкодження, а лише деякі з них. Цей варіант перевірки цілісності відео файлів не є універсальним, адже може виявити пошкодження, які призвели до неспецифічного бітового потоку. Саме це і виявляє дешифратор.

Цілком можливо, що файл був пошкоджений таким чином, що його помилки не призводять до неспецифічного бітового потоку, але все ще не дає зрозуміти, що у файлі, наприклад, мають місце візуальні артефакти.

Наразі достовірно невідомо, чи існує спосіб автоматичного виявлення таких помилок. Найбільша проблема в тому, що для виявлення помилок потрібно знати, як виглядає файл, та що він собою представляє без пошкодження. Це потрібно для порівняння та можливого усунення цих пошкоджень відео файлів.

Наступною проблемою при перевірці цілісності відео файлів є наявність неповних відео файлів, обрізаних, тощо. Можливість виявлення неповних / усічених файлів сильно залежить від індивідуального формату файлу.

З іншого боку, формати, такі як MP4, мають глобальну структуру даних з точними положеннями всіх кадрів / пакетів, тож, якщо чогось не вистачає, то це дуже легко виявити.

Опис процесу виявлення помилок за допомогою HBBatchBeast. HandBrake – інструмент для конвертування відео у майже будь який формат, використовуючи великий набір найновіших існуючих кодеків (рис. 2.1). Це безкоштовний додаток для всіх платформ (Windows, MacOS, Linux) [14].

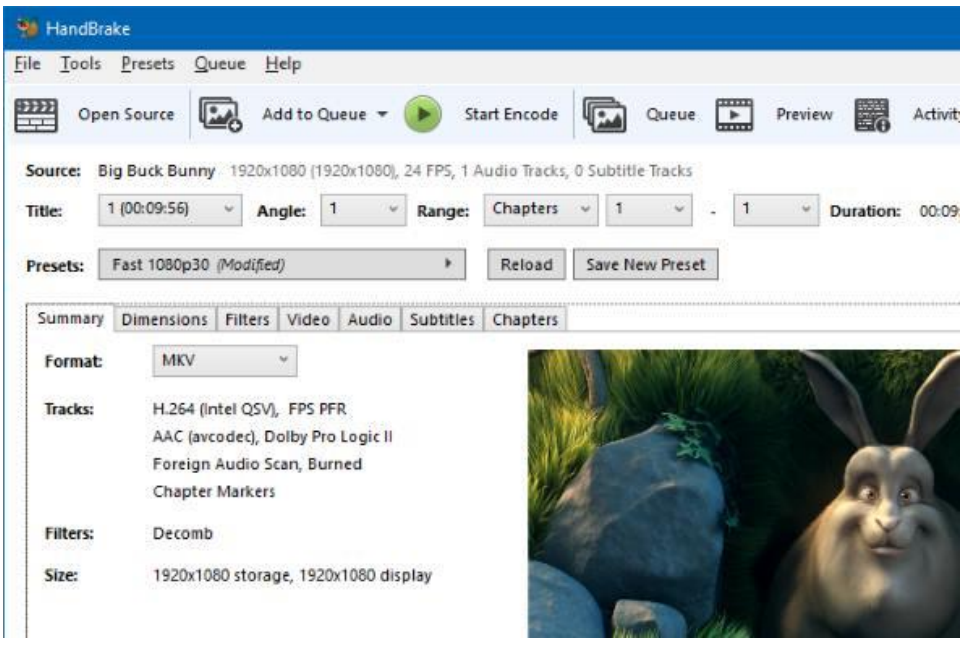


Рис. 2.1 – Скріншот інтерфейсу додатку HandBrake

HBBatchBeast – це безкоштовний додаток для HandBrake та FFmpeg / FFprobe для Windows, macOS та Linux з акцентом на багаторазове перетворення пакетного екземпляра Handbrake (включаючи рекурсивні сканування папок та перегляд папок).

Структура папки призначення зберігається такою ж, як і структура вихідної папки. Медіа в папках також конвертуються. Для кожної папки можна відстежувати декілька папок і встановлювати різні пресети перетворення. Існує також функція перевірки стану здоров'я, яка може сканувати пошкоджені відео файли за допомогою функції "--scan" Handbrakes, хоча це не завжди точно. Це автономна програма в Windows, але вимагає встановлення Handbrake на Linux та Mac.

Цей додаток створений для перегляду папок і файлів та рекурсивного перекодування папок. У програмі є вікно, куди ви можете ввести власну задану програму, проте, якщо ви просто введете "--scan", програма не буде робити перекодування, а просто оцінить, чи здоровий файл чи ні - якщо він пошкоджений тоді він видасть помилку, і вона покаже вам інтерфейс. Нижче наведені два рисунки, що показують інтерфейс налаштувань та результати сканування.

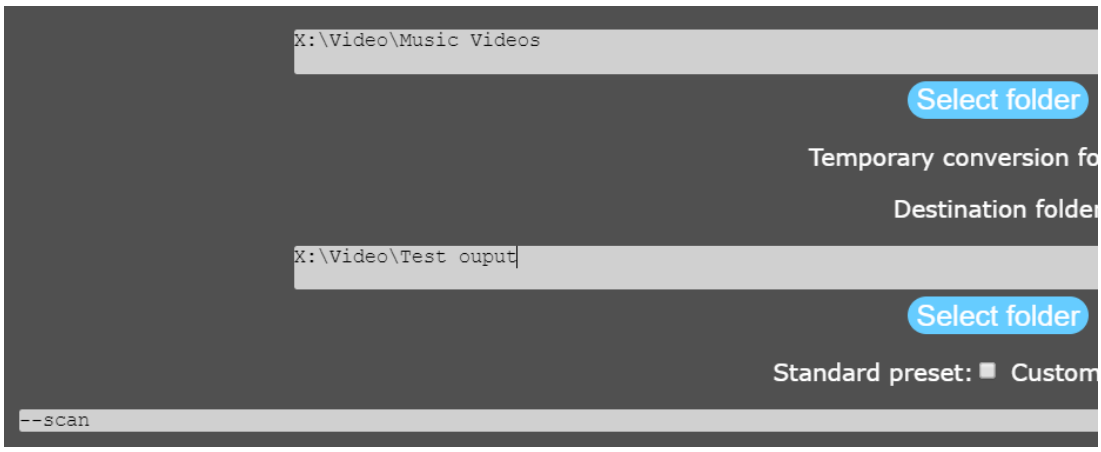


Рис. 2.2 Інтерфейс налаштування для сканування відео файлу

	--scan	Completed
	--scan	Completed
	--scan	Completed
	--scan	Completed
	--scan	Error

Рис. 2.3 Перегляд помилок

Цей додаток дуже швидко сканує бібліотеку відео фалів – сканування більше ніж 2000 відео файлів відбувається менше ніж за 15 хвилин і виявило купу пошкоджених файлів. Варто зазначити, що це сканування також не є стовідсотково точним, але набагато краще, ніж нічого. Також недоліком додатку є те, що програма виявить помилку у відео файлі, якщо назва файлу містить спеціальні символи, такі як "%", або символи з наголосами.

2.6. Метод щілинного сканування відео файлів для перевірки цілісності відео файлів

Поняття щілинної зйомки (сканування). Залежно від геометрії побудови знімка можна виділити три види фотозйомки: кадрова, скануюча, щілинна.

Кадрова. При кадровій зйомці усі точки кадру фіксуються в один момент часу (при спрацюванні затвора). Такий знімок має єдиний центр проекції та строгу геометрію побудови зображення. Схему побудови кадрової зйомки показано на рис. 2.4.

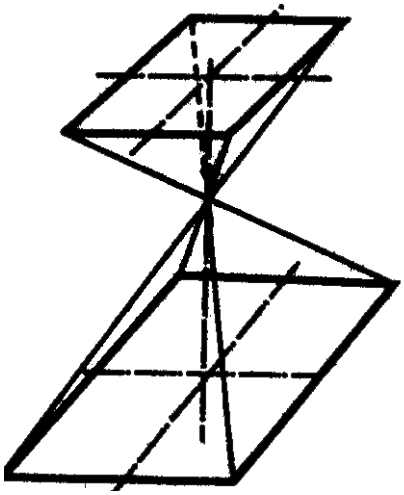


Рис. 2.4 Схема отримання кадрової зйомки

Головним недоліком такого методу є недостатня оперативність, адже потрібно чекати, поки буде знято і зафіксовано на носієві інформації все зображення. Для створення панорамних знімків з різних зон, знятих окремо, ці окремі кадри потрібно точно поєднувати, що потребує досить багато часу.

Скануюча. У скануючих камерах побудова зображення як по рядку, так і по кадру відбувається попіксельно, тобто кожна точка фіксується в окремий момент часу та має свій центр проекції [15]. Схему побудови скануючої зйомки показано нижче на рис. 2.5.

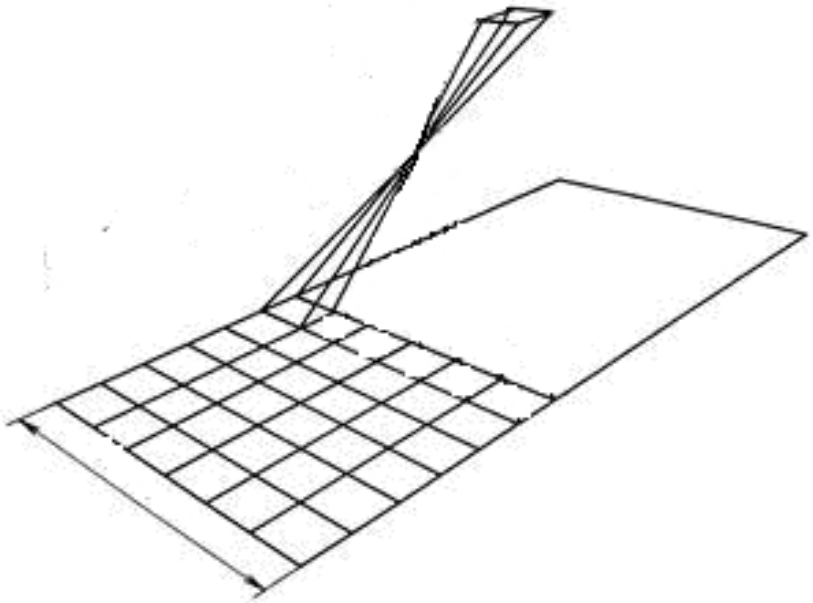


Рис. 2.5 - Схема отримання скануючої фотографії

За своїми геометричними властивостями скануючий знімок, що складається з окремих елементів, поступається кадровому. Однак, скануюча зйомка, на відміну від кадрової, має великі можливості завдяки використанню вузьких знімальних зон для отримання зображення в широких спектральних діапазонах. Крім того, вона забезпечує швидку передачу інформації та можливість подання знімка в цифровому вигляді, що дозволяє використовувати комп'ютерні технології для його тематичної обробки.

Щілинна. При щілинній зйомці зображення отримують послідовно рядок за рядком (рис. 2.6). У площині зображення встановлюється непрозорий екран з вузькою щілиною, через яку зображення реєструється на фотоплівці. Роль щілини може також відігравати лінійка фотоприймачів, що перетворює в електричний сигнал один рядок в просторі предметів [16]. Сканування простору предметів проекцією експозиційної щілини (або проекцією лінійки фотоприймачів) здійснюється переміщенням знімальної камери уздовж координати X.

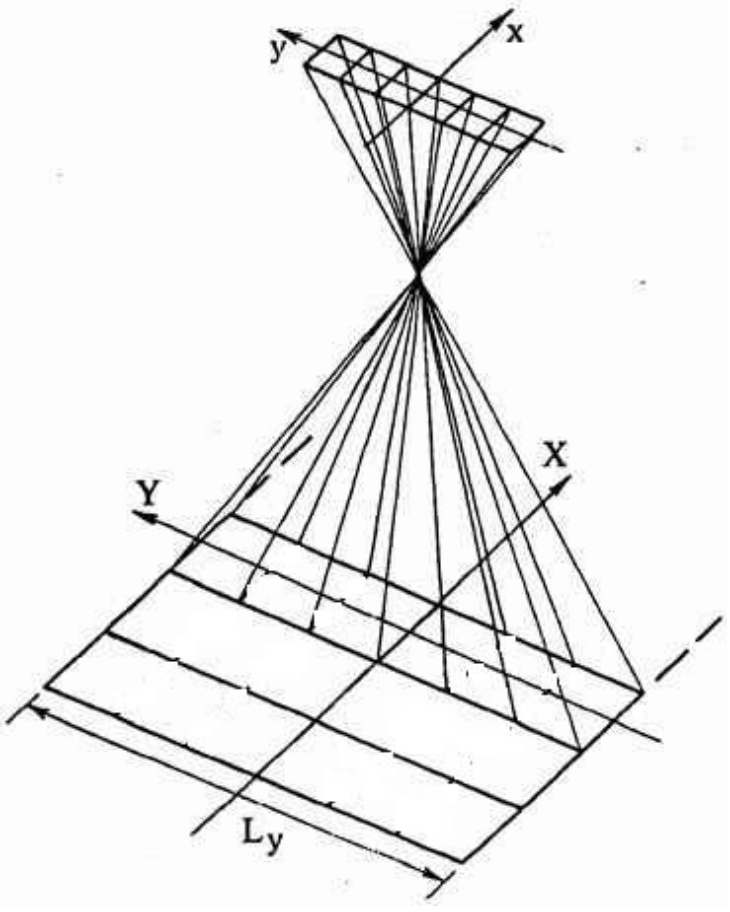


Рис. 2.6 - Схема отримання щілинної фотографії

Щілинна камера робить фотографії в піксель шириною, а потім склеює вертикальні смужки у цілу картину. В результаті ми бачимо, що відбувалося у фокусі щілини протягом певного часу. Також створення зображення може здійснюватися за рахунок переміщення об’єкта зйомки відноносно фотокамери, що відповідає створенню ширококутних панорам. При фотографуванні таким методом рухомих об’єктів кінцевим результатом буде зображення, де все нерухоме – розмите, а те, що рухоме, відображається у вигляді, наближеному до звичного, проте містить цікаві аномалії.

Як було згадано раніше, знімки, що відповідають методу щілинної зйомки, можна отримати за допомогою спеціальних щілинних камер, які не виробляються серійно. Однак така техніка створення щілинних знімків потребує спеціальних вимог, які можуть бути важко реалізовані.

Існують програмні додатки „Slit-Scan Movie Maker“ (рис. 2.7) для MAC OS [17], програма „After Effects“ від Adobe [18] та інші, що дозволяють отримувати ефекти, які схожі на ефекти, що отримуються методом щілинної зйомки.

Рис. 2.7 Головне вікно програми Slit-Scan Movie Maker

After Effects це програмний додаток від Adobe (рис. 2.8), який дає змогу розробляти вражаючі анімовані ефекти. Ця програма використовує графічну

обробку зображення, не використовуючи основні принципи щілинної зйомки. Однак ці програми є платними і входять до складу цілого програмного комплексу, що не є зручним при використанні.

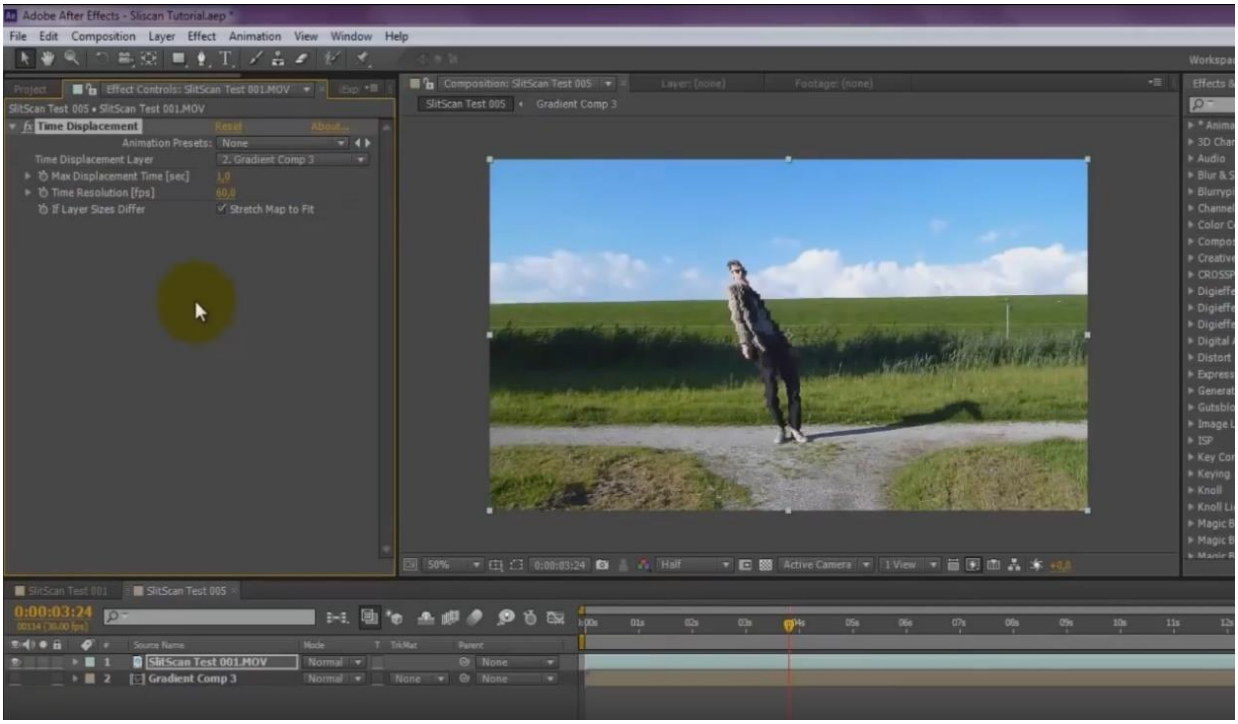


Рис. 2.8 Інтерфейс додатку After Effects

Також існують розважальні додатки для ОС Android та iOS, які використовують камеру пристрою, щоб створювати зображення, що відповідають методу щілинної зйомки. Їх головний недолік – це недостатня функціональність та невелика кількість налаштувань. Схожі ефекти можна отримати за допомогою програми „VideoCube“ [19] від Microsoft Research, яка, нажаль, підтримує лише формат нестиснутих відео файлів і застосовує інші поняття та параметри налаштування, відмінні від методу щілинної зйомки. Інтерфейс програмного засобу показаний на рис. 2.9.

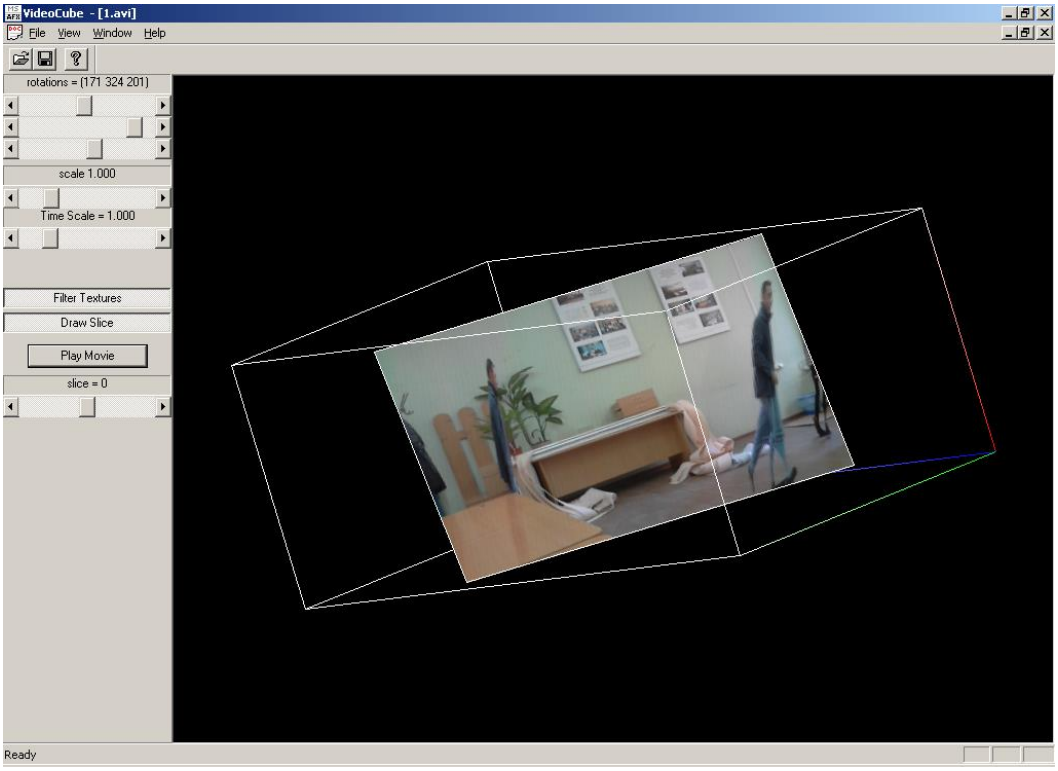


Рис. 2.9 Інтерфейс VideoCube

Виходячи з принципів отримання щілинного знімка, нескладно визначити методи його цифрового моделювання. Пропонується замість механічного способу створення щілинних фотографії використовувати програмну обробку відзнятого, з великою кількістю кадрів в секунду, відео. Нестача кадрів в секунду призведе до часткової пікселізації фотографії, надлишок – тільки до зайвої ширини одержуваної фотографії, однак це легко може бути усунено в будь-якому редакторі зображень.

Схематично наші міркування стосовно обробки відео для отримання цифрових фотографій, що аналогічні методу щілинної зйомки, представлені на рис. 2.10.

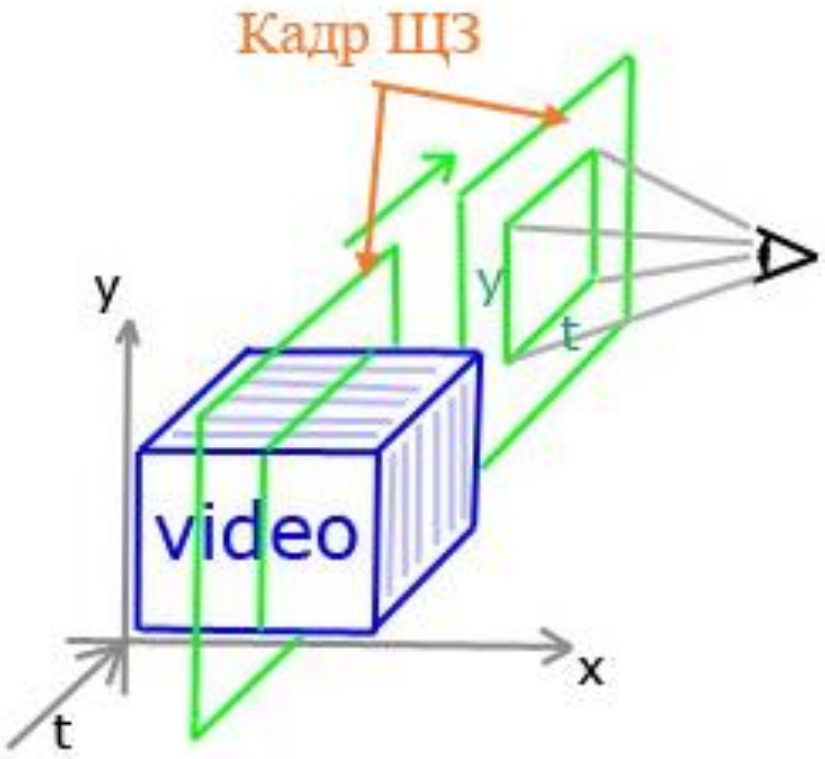


Рис. 2.10 - Схема отримання фото щілинної зйомки з відеоряду

Висновки до розділу 2

В розділі було розглянуть теоретичну інформацію про поняття дешифрування та декодеру. Для того, щоб перевірити відео файли на пошкодження зазвичай використовують дешифратори. Проте такий процес є складним та потребує теоретичні знання з дешифрування та знання деяких фреймворків. Спеціальних програмних засобів для перевірки відео файлів на цілісність не існує.

Для того щоб дізнатися чи є у файлі пошкодженні кадри, звісно, його можна просто передивитися у будь-якому відео-плеєрі. Проте, чи можливо це зробити за короткий час, якщо відео файл – це повноцінний фільм на більш ніж дві години. Саме тому пропонується використовувати щілинну зйомку для обробки відео як спосіб перевірки цілісності відео файлів. За допомогою сканування відео можна передивитися усі кадри за лічені секунди, адже фотографії, створені методом щілинної зйомки містять у собі інформацію про кожний кадр оброблюваного відео.

У розділі також було ретельно розглянуто різні формати відео файлів та обрано найпоширеніший для подальшої роботи над перевіркою цілісності даних відео.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Цифрове моделювання ефектів щілинної зйомки

Для дослідження відео файлів за допомогою щілинної зйомки в середовищі Borland Delphi 7 [20] розроблено програмний засіб, що дозволяє власноруч отримувати фотографії, створені таким методом (рис. 3.1). Програмний засіб надає змогу створити фото з різними налаштуваннями. У налаштуваннях програми є можливість виставити позицію щілини, швидкість руху щілини, обирати горизонтальну чи вертикальну щілинну зйомку або статичну чи рухому. Одним з головних параметрів обробки відео є вибір початкового та останнього кадру відео, які будуть оброблятися. Отримані за допомогою додатку кадри зберігаються вигляді файлів BMP. Усі додаткові залежності і отримані, в залежності від параметрів, ефекти обробки будуть описані далі.

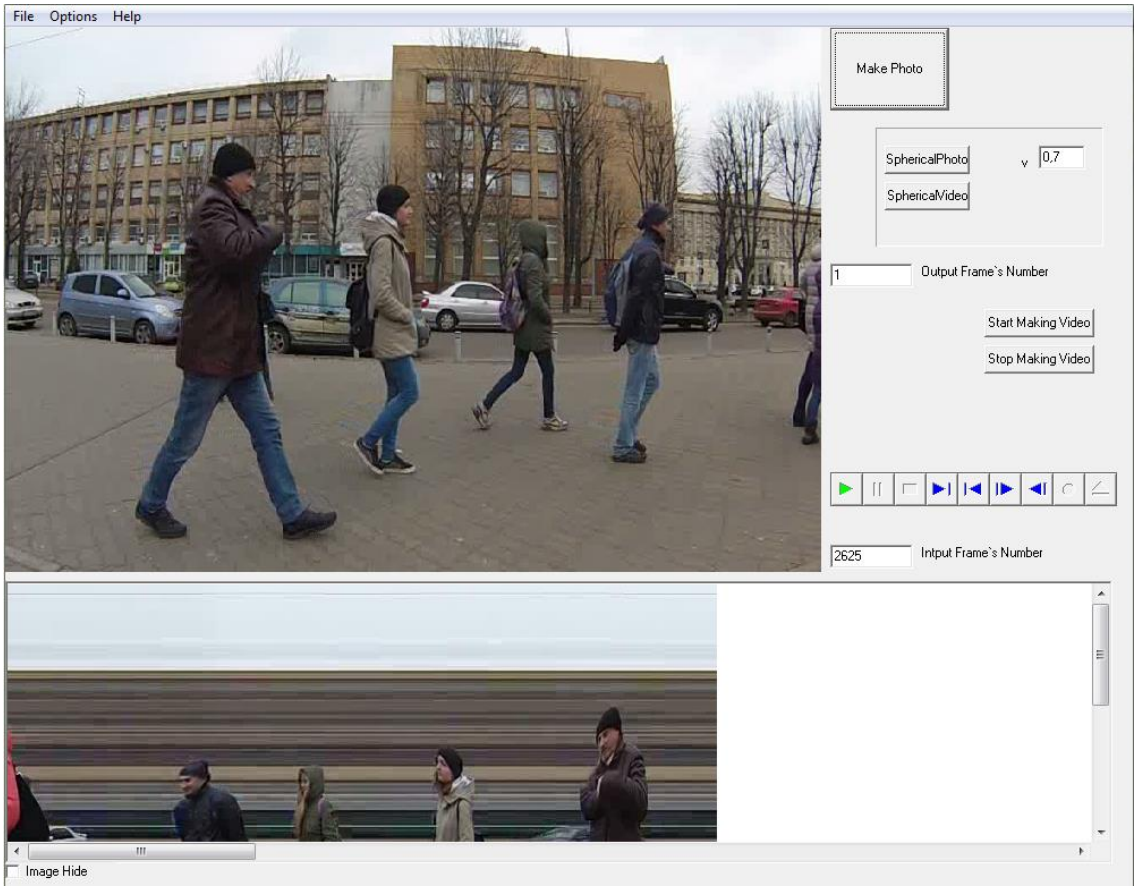


Рис. 3.1 - Головне вікно програми

Для моделювання щілинної зйомки з вертикальною щілиною один стовпець пікселів, що відповідає положенню віртуальної щілини, витягується з кожного оригінального кадру відео і розміщуються послідовно один за одним в порядку збільшення індексу (горизонтальної координати) стовпця. Пояснення методики і ефектів, отриманих за її допомогою , представлено на рис. 3.2.

Програмно на мові Object Pascal процес створення просторово-часового кадру з нерухомою щілиною може бути реалізований наступним чином:

```
for i:=Frame_start to Frame_last do
 begin
  InputFrame.Position:=i;
   for j:=1 to InputImage.Height do
    OutpuImage.Canvas.Pixels[i-
Frame_start,j]:=InputImage.Canvas.Pixels[Slit_Position,j];
 end;
```

де Frame_start – початковий кадр відео

Frame_last – кінцевий кадр відео

Slit_Position – позиція щілини.

На рис. 3.3 показані фото, отримані даним алгоритмом. Для обробки використовувалось відео, зняте автором на людних вулицях міста Черкаси.

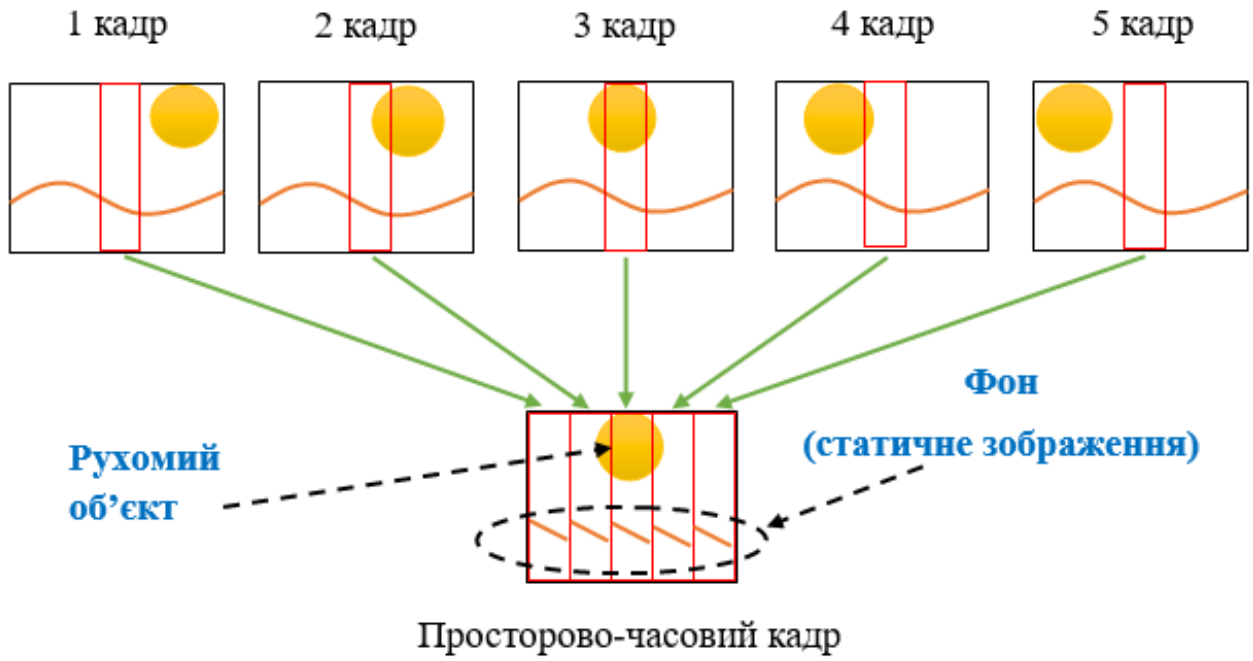


Рис. 3.2 Схема пояснення ефекту на фотографіях, створених методом щілинної зйомки з вертикальною статичною щілиною.

Оскільки кожний кадр оброблюваного відео представляє собою різні моменти часу, отримуване зображення – це зображення подій, що відбулися на протязі певного часу. Довжина зображення дорівнює кількості кадрів, що були оброблені. статичні (нерухомі) предмети відображаються смужками (фон), рухомі об'єкти з'являються у вигляді, близькому до оригінального.

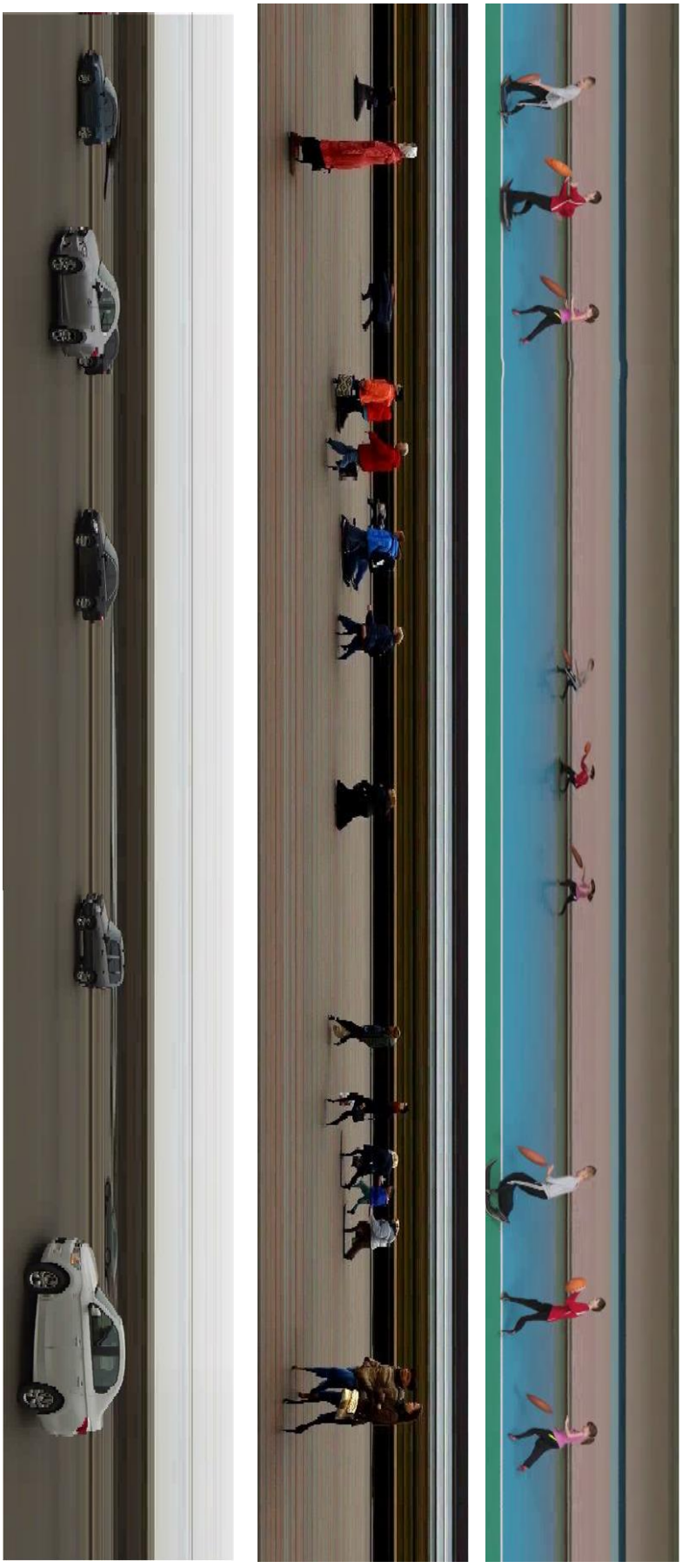


Рис. 3.3 Фотографії, створені за допомогою вертикальної статичної щілинної зйомки

При створенні фотографій методом щілинної зйомки із вертикальною статичною щілиною з кожного кадру оброблюваного відео витягується один вертикальний стовпець пікселів, що відповідає віртуальній щілині, але для кожного кадру положення щілини має різні значення і залежить від швидкості руху щілини. Наприклад, якщо швидкість руху щілини рівна 1, то стовпчик для наступного кадру береться на одну позицію лівіше, ніж з попереднього кадру і так для усіх кадрів (рис. 3.4).

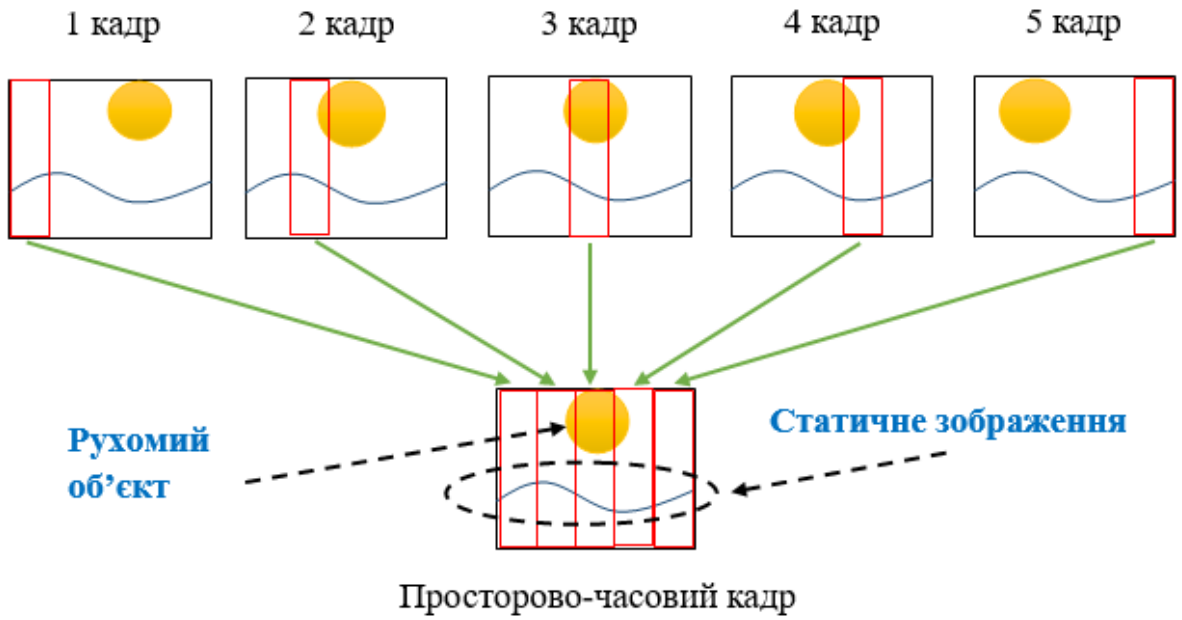


Рис. 3.4 Схема пояснення ефекту на фотографіях, створених методом щілинної зйомки з вертикальною рухомою щілиною

Програмно на мові Object Pascal процес створення просторово-часового кадру з рухомою щілиною може бути реалізований наступним чином:

```
for i:=Frame_start to Frame_last do
 begin
  InputFrame.Position:=i;
  Slit_Position:=Slit_Position+V;
  for j:=1 to InputImage.Height do
   OutpuImage.Canvas.Pixels[i
Frame_start,j]:=InputImage.Canvas.Pixels[Slit_Position,j];
  end;
```

де Frame_start – початковий кадр відео

Frame_last – кінцевий кадр відео

Slit_Position – позиція щілини.

Цікавою особливістю щілинної зйомки із вертикальною рухомою щілиною є те, що на фотографіях з'являється зображення нерухомих предметів, які в статичній щілинної зйомки відображались смугами. Величина ефекту залежить від швидкості зміни положень щілини в двох сусідніх кадрах - швидкості руху щілини. Якщо V=1 (рис. 3.5), то довжина статичного предмету відповідає оригінальній, якщо 0<V<1 (рис 3.6 та рис 3.7), то статичні предмети на фотографії витягуються, проте рухомі об'єкти залишаються незмінними (додаток Х), де V – швидкість руху щілини.

Рис. 3.5 Швидкість руху щілини V=1

Попередній рисунок – це фотографія отримана за допомогою розробленого програмного засобу, використовуючи метод рухомої вертикальної щілини. Швидкість руху щілини на цьому зображенні – 1, тобто кожний наступний стовбець пікселів береться послідовно, кадр за кадром і ширина отриманого зображення є рівна ширині відео, що сканувалося.

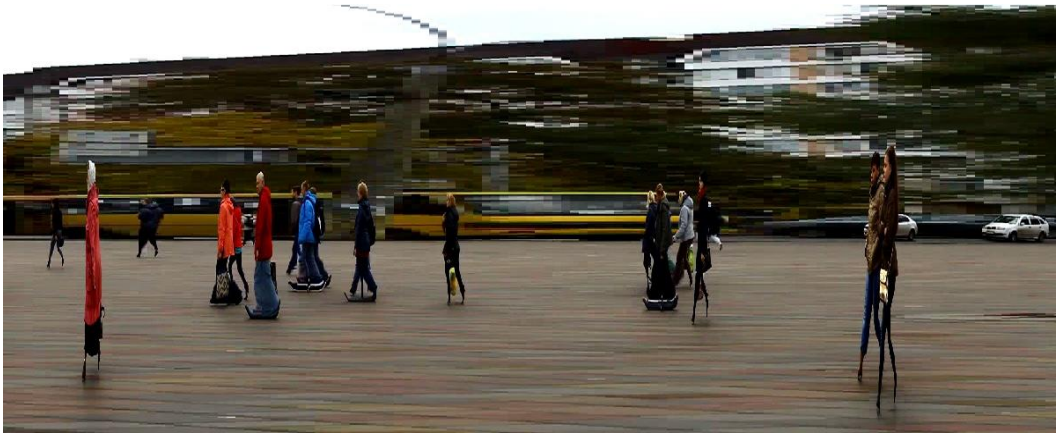


Рис. 3.6 Швидкість руху щілини V=0,1

Вище представлено зображення (див. рис. 3.6), отримане шляхом обробки того ж самого відео, що і попереднє, методом з рухомою вертикальною щілинию. Швидкість руху була зменшена, тобто статична

частина зображення подекуди дублювалась. Зображення вийшло ширшим, в порівнянні з попереднім, проте відео фрагмент був однаковий. На рис. 3.7 представлено обробку цього ж відео, проте швидкість руху щілини становила 0,5, тому зображення вийшло не таким широким, як при швидкості 0,1.

Рис. 3.7 Швидкість руху щілини V=0,5

Для моделювання щілинної зйомки з горизонтальною рухомою щілиною виконуються ті ж самі операції, що і при щілинної зйомки з вертикальною рухомою щілиною, але замість стовпчиків використовуються рядки, які знов представляють собою аналог щілини.

Не зважаючи на те, що така методика отримання фотографій схожа на попередні, проте ефект буде носити зовсім інший характер.

За допомогою цього методу можна отримати цікаві ефекти при зйомці рухомих об’єктів, що обертаються навколо осі, коли предмет рухається з перемінною швидкістю або якщо частини об’єкта рухаються із змінними швидкостями (рис. 3.8).

На рис. 3.9 представлено пояснення ефектів, що отримуються при обробці відео таким способом. Зазвичай такий ефект будують на основі послідовно знятих кадрів статичного предмету, знятого під різними кутами до площини камери. Предмет фотографується через певний крок по куту, а потім фотографії склеюються в фото редакторі з використанням фільтру згладжування. Такий процес займає доволі багато часу. Щілинна зйомка може

дати той самий ефект, але за набагато менший час при меншій технічній складності.

Рис. 3.8 Фотографії предметів, які обертались при щілинній зйомці з горизонтальною рухомою щілиною

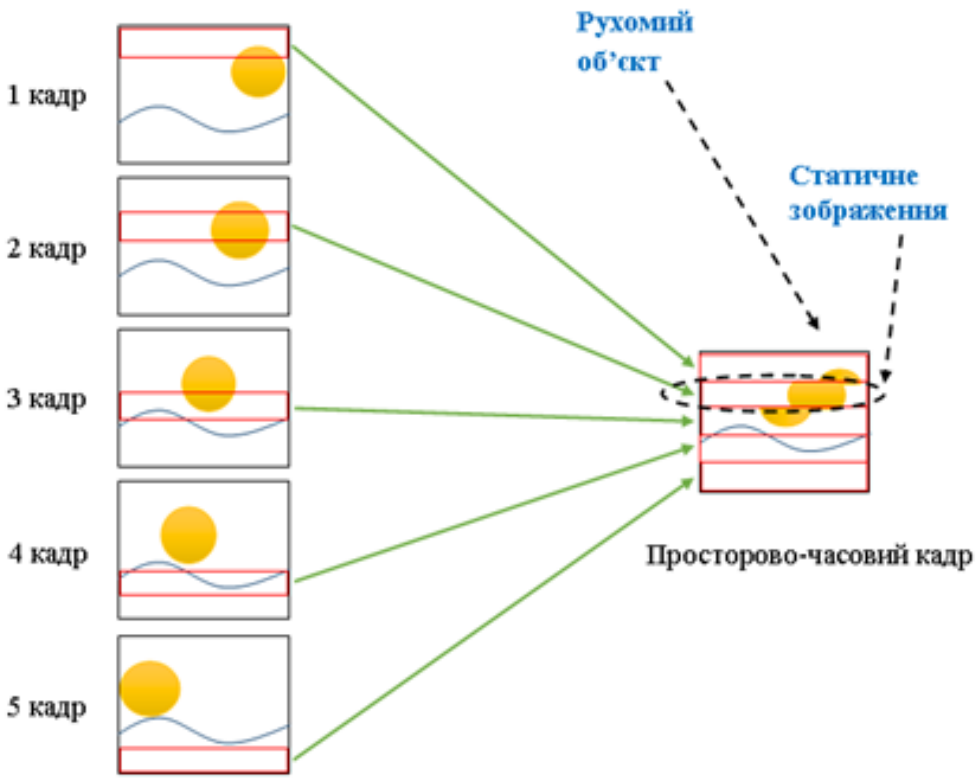


Рис. 3.9 Схема пояснення ефекту на фотографіях, створених методом щілинної зйомки, з горизонтальною рухомою щілиною: статичне оригінальне зображення не змінюється, рухомі об'єкти змінюють свою форму, але не змінюють уявлення про них

3.2. Опис алгоритму виявлення пошкоджень у відео файлі

Традиційний підхід до вирішення проблеми видалення дефектів (пошкоджень) базується на припущенні, що артефакт присутній лише на одному кадрі. Тобто, якщо деякі пікселі не виявлені на попередньому або наступному кадрі – то цей піксель належить до дефекту. Розроблений метод обробки відео та його сканування методом щілинної зйомки заснований на застосуванні просторових, часових або просторово-часових алгоритмів обробки кадру.

Просторовий метод заснований на припущенні, що артефакт має невелику площину. Алгоритм використовує деякі операції математичної морфології [21]. Дефектами на кадрах буде вважатися різниця величин, яке перевищує фіксовану деяку величину. При малих значеннях цієї величини, можливе виявлення помилкових спрацювань, що викликані шумом. Перевагою даного методу є простота у обчисленні. Основний недолік – неможливість виявлення артефактів великого розміру та погано виражених темних та світлих плямах.

Часовий метод заснований на припущенні, що абсолютна різниця пошкодженого пікселя з пікселями з попереднього та наступного кадрів перевищує задане порогове значення, тоді як різниці інтенсивності пікселя перевищують інтенсивність сусіднього пікселя. Цей алгоритм дозволяє доволі якісно діяти у ситуаціях, коли об’єкт перекривається іншим чи раптово зникає з кадру, адже різниця між цими кадрами буде великою тільки в одному часовому напрямку [22]. Недоліком цього алгоритму є те, що не враховується рух об’єктів у камері та між кадрами.

Просторово-часовий метод заснований на аналізі послідовного рядка\стовбця з n пікселів, що взяті з трьох послідовних кадрів .

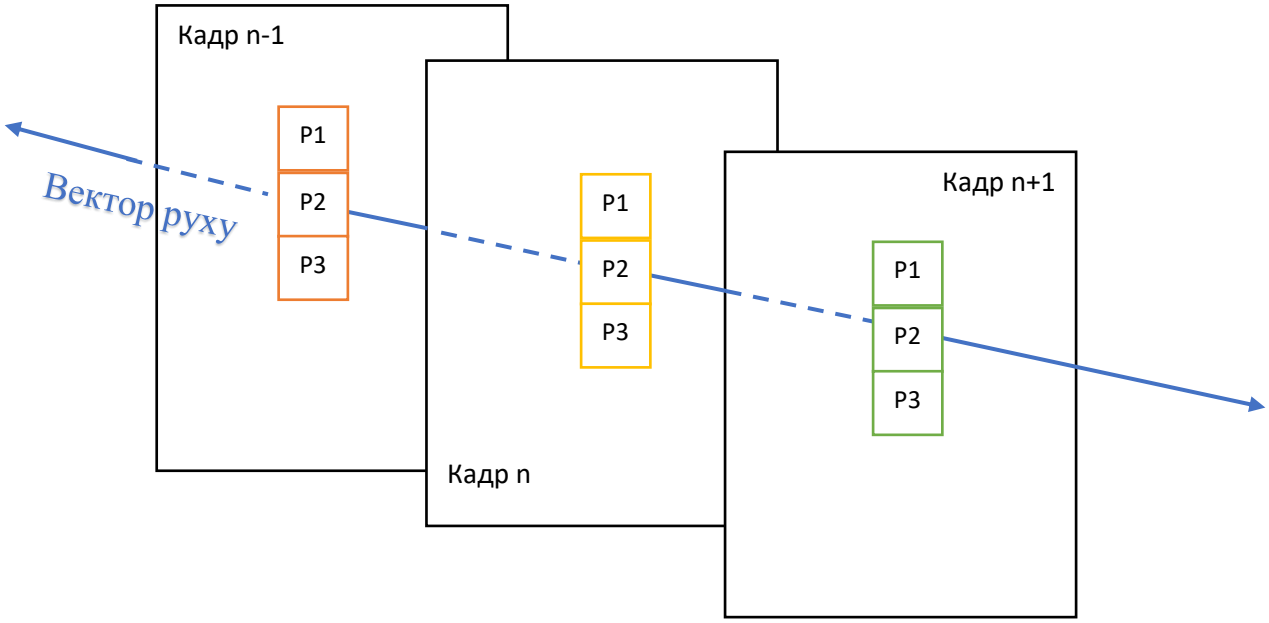


Рис. 3.10. Схема просторово-часового алгоритму

Основний метод перевірки полягає в тому, що дефектні пікселі дуже відрізняються по яскравості та інтенсивності у загальному розподілені яскравості зображення. Для роботи алгоритму сканується перший кадр відео та записується стовпчик пікселів із заданими початковими координатами відносно осі X або У. Параметри кольору кожного з пікселів в отриманому ряді пікселів записується у відповідний масив даних (матрицю). Потім з кожного наступного кадру зчитується та записується та сама інформація, координати по осі X або У не змінюються, змінюється значення часу, тобто матриця даних поповнюється. Потів відбувається порівняння кожного стовпця пікселів із сусідніми. З допусканням невеликих відхилень програма перевіряє ідентичність отриманих сусідніх стовпців, тобто порівнюються стовпці n-1, n та n+1, потім n, n+1 та n+2, тощо. Якщо усі значення відповідають заданим параметрам – на відео не виявлено пошкоджень. Якщо навпаки – програма відтворює на головному вікні отримане зображення з'єднаних послідовно рядів пікселів аби можна було побачити пошкоджені кадри.

У випадку відсутності пошкодження можливий той факт, що артефакти не займали увесь кадр, а лише на деяких невеликих частинах кадру було помітно візуальне пошкодження пікселів. Тому після вдалого сканування відео, потрібно ще раз впевнитися що увесь кадр цілий. Тому алгоритм сканування відео файлів було дороблено. Щоб уникнути невиявлених артефактів на зображенні пропонується зробити декілька сканувань відео із зміною початкової координати. Таких сканувань можна робити безліч, обмежуючись лише роздільною здатністю відео файлу, що сканується.

3.3. Порівняння та аналіз отриманих результатів

За допомогою розробленого програмного засобу було перевірено та проаналізовано більш ніж 30 відео фрагментів. Розробка виконувалась у середовищі Borland Delphi 7. Було спеціально створено та визначено типи

пошкоджень відео файлів. Для обробки використовували файли медіа контейнеру Xvid, розширення було обране .avi. Зображення, що створюються зберігаються у форматі .bmp. Всі відео були відзняті автором на власну камеру, відредаговані та перекодовані, також деякі відео були взяті у вільному доступі у мережі Інтернет.

Отримані результати (рис.3.11) дають змогу дуже якісно відстежити та визначити пошкодження та порушення цілісності відео файлів. Отримані якісні зображення на яких можна порівнювати оригінальне та пошкоджене відео. Усі пояснення та види знайдених порушень цілісності відео описані далі.

Рис. 3.11 Виявлення пошкоджених кадрів у відео

Через пошкодження відео, тобто псуванням комп’ютерних даних, що було викликане людським чи апаратним фактором. Зазвичай саме такі пошкодження не можуть з’явитися самі по собі. Коли відео пересилається, передається чи навіть проходить через процес архівації воно може бути пошкодженим. Особливо коли відбувається зчитування та копіювання файлів із зовнішнього носія даних (CD, DVD-диску або USB-флешки).

Такі пошкодження можуть бути викликані аварійним завершення роботи комп’ютера. Також це може виникнути через неправильний запис відео на

електронний носій: у випадку, коли не до кінця був завершений процес копіювання чи запису відео.

Також через фізичні дефекти електронного або зовнішнього носія інформації може також виявитись після запису на цей носій, що відео має пошкоджені кадри. Саме такі кадри дуже легко визначити та проаналізувати у розробленому програмному додатку. Найрозповсюдженою проблемою є запис відео на жорсткий диск, у якому наявні «биті» сектори. Задля порівняння та аналізу роботи програми був створений короткий відеоматеріал із статичним фоном та пішоходами, що проходять паралельно відносно площини зйомки. Це порівняння потрібно для визначення всіх можливих помилкових виявлень пошкоджень відео.

Потім у відео редакторі було навмисно пошкоджено деякі фрагменти відео. Як результат сканування обох відео фрагментів, програма не виявила пошкоджень на першому відео, проте вказала на істотні помилки при скануванні більш ніж 40 кадрів серед 600 кадрів у відео. На рис. 3.11 та рис. 3.12 можна побачити порівняння оригінального та пошкодженого відео за допомогою щілинної зйомки та розробленого програмного засобу.

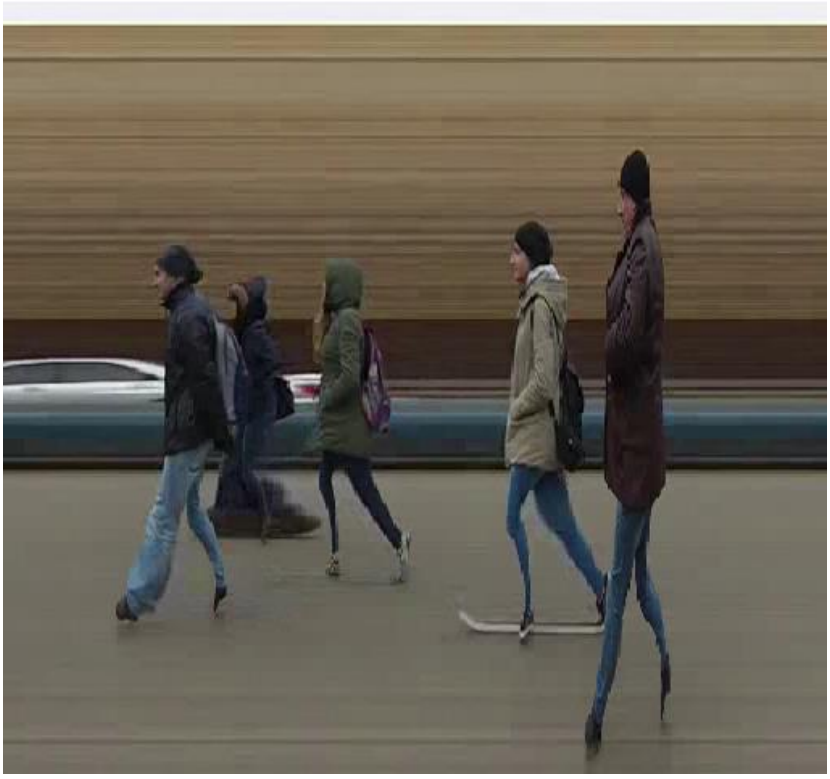


Рис. 3.11 Результат сканування оригінального відео

У результаті, сканування оригінального відео фрагменту виявило пошкодження на тих ділянках відео файлу, де проходили повз люди, а особливо на фрагментах «проходу» людини повз щілину. Аби надалі уникати подібних помилок, використовується додаткова перевірка різниці яскравості та інтенсивності пікселів, адже зазвичай, переважна більшість артефактів значно відрізняється за цими параметрами.

На рис. 3.12 представлено зображення, отримане після сканування того самого відео фрагменту, проте вже з порушенням його цілісності та якості.

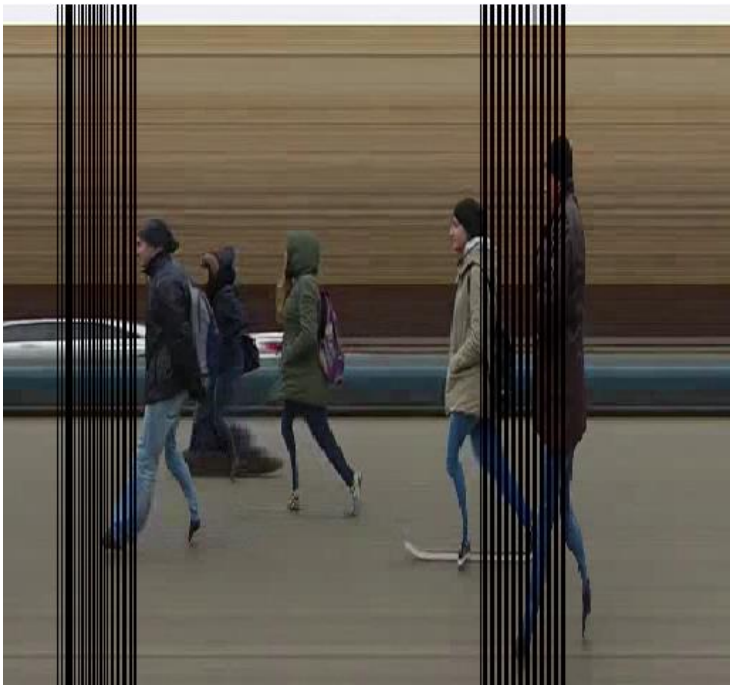


Рис. 3.12 Результат сканування пошкодженого відео

Як результат бачимо, що програма виявила пошкодження кадрів (див. рис. 3.12). Ці кадри під час неправильного запису були витраченими, тому на отриманому зображені ми бачимо чорні смужки. Ці смужки вказують, що кадри, повз які пройшло сканування виявились відсутніми або пошкодженими.

Також, аналізуючи можливі пошкодження кадрів важливим фактором є роздільна здатність відео файлу. На отриманих зображеннях можна спостерігати часткову пікселізацію та відсутність чіткості зображення. Це зумовлено насамперед можливостями відео камери, на яку було знято дані

відео фрагменти. Основною проблемою, що може виникати при скануванні неякісного відео файлу може стати виявлення помилки відсутності кадрів, адже при низькому значенні параметру кількості кадрів в секунду отримуване відео може відображатись ривками. Тоді алгоритм визначає це як помилку відсутності кадрів, викликана низькою частотою кадрів, а не через пошкодження його цілісності. На рис. 3.13 зображено виявлення значних пошкоджень відео файлу.

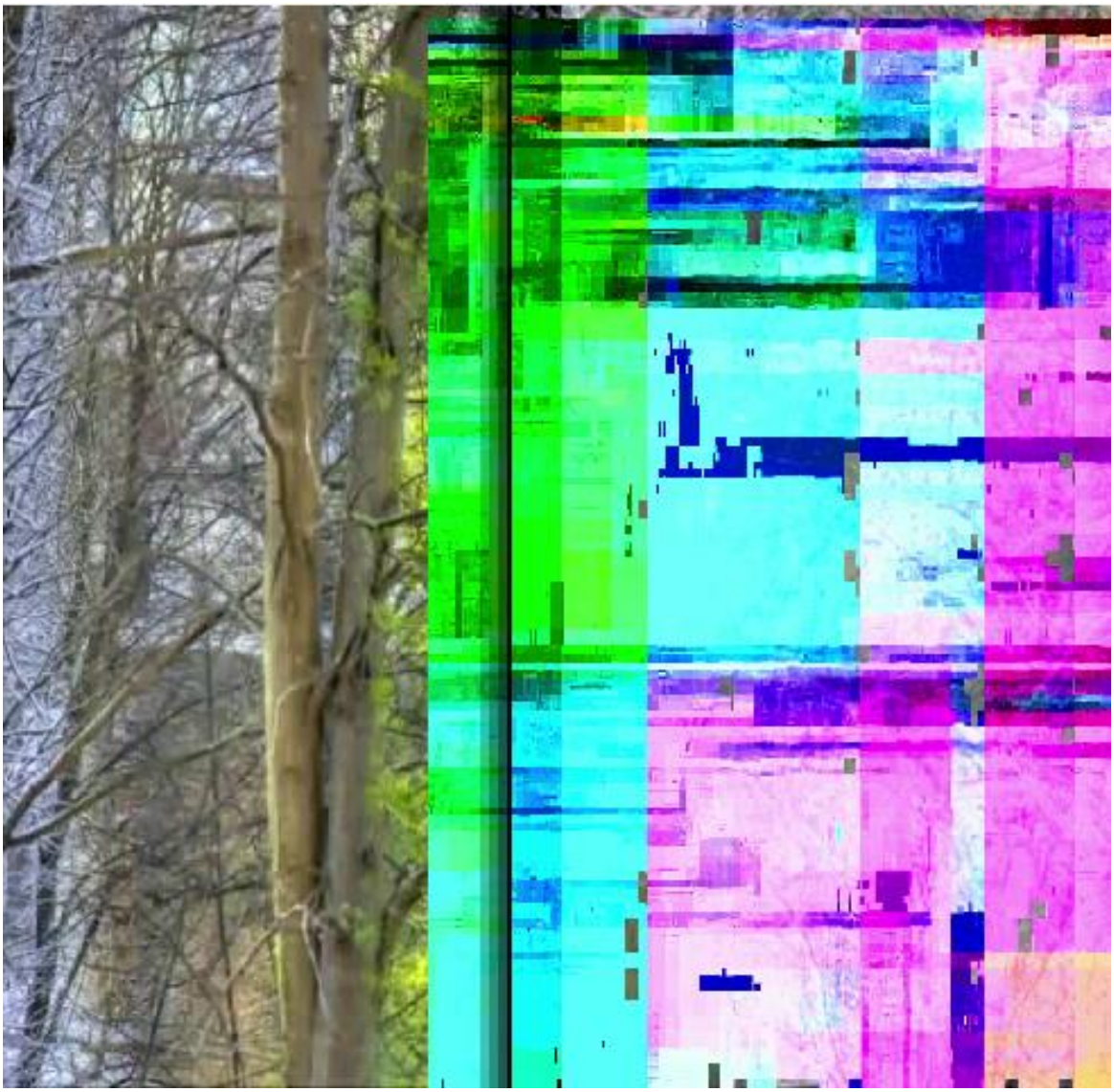


Рис. 3.13 Значне пошкодження відео файлу, виявлене за допомогою сканування із рухомою вертикальною щілиною

Особливістю розробленої програми є те, що за дуже малий час (1-2 секунди) вона може проаналізувати весь відеоряд та визначити, що файл є

пошкодженим або неповним. Підручними засобами це неможливо виявити без перегляду самого відео файлу. Адже відео файл, наприклад може бути двох часовим чи навіть більше. Ще можна зазначити, що звичайної перевірки властивостей файлу не дасть змоги виявити порушення цілісності. Адже під час досліджень було виявлено, що при навмисному пошкодженні відео файлів їх розмір, роздільна здатність, формат та тривалість абсолютно не змінюються.

Висновки до розділу 3

При розробці даного розділу було створено програмний додаток для обробки відео та сканування його кадрів методом щілинної зйомки.

Було описано декілька методів щілинної зйомки та проілюстровано схему їх роботи. У розділі також представлено ілюстрації порівняння пошкоджених відео файлів та оригінальних. Створений алгоритм перевірки цілісності відео файлів полягає в тому, що відео розбивається на кадри, для того щоб проаналізувати кожний з них. Потім порівнюється значення яскравості та інтенсивності кожного рядка чи стовпця пікселів, отриманих після сканування. З допусканням невеликих відхилень програма перевіряє ідентичність отриманих сусідніх стовпців. Якщо усі значення відповідають певним заданим параметрам, то на відео немає пошкоджень.

Було проведено аналіз та сканування відео файлів на різні пошкодження. В результаті було проаналізовано найпоширеніші помилки при скануванні файлів, було виявлено відсутність кадрів, заміну кольору пікселів. Розроблений програмний додаток швидко проаналізував та визначив пошкоджені файли.

ВИСНОВКИ

В результаті виконання бакалаврської дипломної роботи було розглянуто поняття цілісності даних, пошкодження даних та перевірки даних на достовірність. Цілісність даних є важливим фактором у будь-якому інформаційному середовищі. Точна та послідовна передача інформації є обов'язковим фактором для правильної роботи системи. Під час зберігання, редагування та операцій обробки даних можуть виникати їх пошкодження в наслідок певних зовнішніх та внутрішніх факторів. Такими факторами є: некоректне або аварійне завершення роботи системи або програми, фізичні дефекти або апаратну не функціональність, наявність шкідливого програмного забезпечення, що приводять до навмисного пошкодження файлів.

Серед існуючих способів перевірки цілісності файлів є перевірка контрольної суми даних. Контрольна сума - це унікальне значення, отримане за допомогою застосування алгоритму хеш-функції. Алгоритмів хешування існує безліч, деякі з найбільш часто вживаними є: MD5, CRC32, SHA-1, SHA256, BTIH. Хеш-функція загалом призначена для згортки вхідного масиву будь-якого розміру в рядок бітів. Вона використовується наприклад для швидкого порівняння двох масивів на рівність: якщо у двох масивів хеші різні, то масиви гарантовано різні, а в разі рівності хеш - масиви швидше за все рівні. Однак найчастіше хеш-функції використовуються для перевірки унікальності пароля, файлу, рядка і тощо. В роботі було проведено порівняння цих алгоритмів.

Для перевірки цілісності відео файлів немає простого способу дізнатися пошкоджений файл чи ні. Більшість форматів відео контейнерів не містять контрольних сум, тому немає ніяких способів перевірити їх достовірність. Все що можна зробити - це спробувати декодувати файл і побачити, чи розшифровує декодер несподівані помилки чи пошкодження.

У роботі було запропоновано метод щілинного сканування відео файлів для перевірки цілісності. Щілинна зйомка представляє собою набір фотографій

в піксель шириною, що потім склеюються у цілу картину. При створені зображень таким методом рухомих об’єктів кінцевим результатом буде зображення, де все нерухоме – розмите, а те, що рухоме, відображається у вигляді, наближеному до звичного.

Для дослідження відео файлів за допомогою щілинної зйомки в середовищі Borland Delphi 7 розроблено програмний засіб, що дозволяє власноруч отримувати фотографії, створені таким методом. Розроблено алгоритм виявлення пошкоджень у відео файлі шляхом сканування його кадрів методом щілинної зйомки. Створений алгоритм перевірки полягає в тому, що відео розбивається на кадри, для того щоб проаналізувати кожний з них. Потім порівнюється значення яскравості та інтенсивності кожного рядка чи стовпця пікселів, отриманих після сканування. З допусканням невеликих відхилень програма перевіряє ідентичність отриманих сусідніх стовпців. Якщо усі значення відповідають певним заданим параметрам, то на відео немає пошкоджень. Якщо навпаки - то відео файл містить певні артефакти чи дефекти. Програмний засіб передбачає помилкові спрацювання та дозволяє додатково перевірити цілісність, шляхом зміни вхідних параметрів початкової координати щілини. У роботі представлено приклади сканування пошкоджених відео фрагментів, які виникають внаслідок різних зовнішніх факторів.

СПИСОК ЛІТЕРАТУРИ

1. Глушков В. Енциклопедія кібернетики. – М.: Нолидж, 1973. – 582 с.
2. Observations on Errors, Corrections, & Trust of Dependent Systems [Електронний ресурс] // James Hamilton – Режим доступу до ресурсу: https://perspectives.mvdirona.com/2012/02/observations-on-errors-corrections-trust-of-dependent-systems/.
3. Щербаков А. Современная компьютерная безопасность / Щербаков., 2009. – 145 с.
4. Хеш-функція [Електронний ресурс] – Режим доступу до ресурсу: http://studcon.org/hesh-funkciya.
5. Куц М. Розробка алгоритму для прискорення порівняння файлів // Інтернаука. – 2016. – №7. – С. 19–21.
6. Md5Checker [Електронний ресурс] – Режим доступу до ресурсу: http://getmd5checker.com/.
7. EF CheckSum Manager [Електронний ресурс]. – Режим доступу до ресурсу: http://www.efsoftware.com/cm/e.htm
8. WinMD5Free [Електронний ресурс] – Режим доступу до ресурсу: https://www.winmd5.com/.
9. CimTrak [Електронний ресурс]. – Режим доступу до ресурсу: https://www.cimcor.com/solutions/servers
10. Use the System File Checker tool to repair missing or corrupted system files [Електронний ресурс] // Windows support – Режим доступу до ресурсу: https://support.microsoft.com/en-us/help/929833/use-the-system-file-checker-tool-to-repair-missing-or-corrupted-system.
11. Навчальний посібник з дисципліни «комп'ютерна схемотехніка та архітектура комп'ютерів» – ПолтНТУ, 2017. – 98 с.
12. Б. Шнайер. Прикладна криптографія. Протоколи, алгоритми, вхідні тексти на мові Сі – М.: Диалектика, 2002 – 610 с.

13. Основні відео формати. Стандарти стиснення і кодеки [Електронний ресурс] – Режим доступу до ресурсу: https://ureader.ru/uk/main-video-formats-compression-standards-and-codecs/.
14. HBBatchBeast [Електронний ресурс] – Режим доступу до ресурсу: http://hbbatchbeast.io/.
15. В. В. Гнатушенко. Моделювання процесу формування цифрових сканерних зображень дистанційного зондування – 2005. – 46 с.
16. Щелевая съёмка / Slit-scan photography [Електронний ресурс] – Режим доступу до ресурсу: https://it2see.livejournal.com/924.html.
17. Slit-Scan Studio [Електронний ресурс] – Режим доступу до ресурсу: https://apps.apple.com/us/app/slit-scan-studio/id493342965?mt=12.
18. After Effects [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/ua/products/aftereffects.html.
19. VideoCube [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/en-us/download/details.aspx?id=52646.
20. Delphi world 7 [Електронний ресурс]. – Режим доступу до ресурсу: http://delphiworld.narod.ru/_all_articles_.html
21. Методы удаления дефектов в видео [Електронний ресурс] / Віктор Шелудько – Режим доступу до ресурсу: https://it2see.livejournal.com/924.html.
22. Вірт Н. Алгоритми та структури даних. – М: Діалектика, 1989. – 360 с.

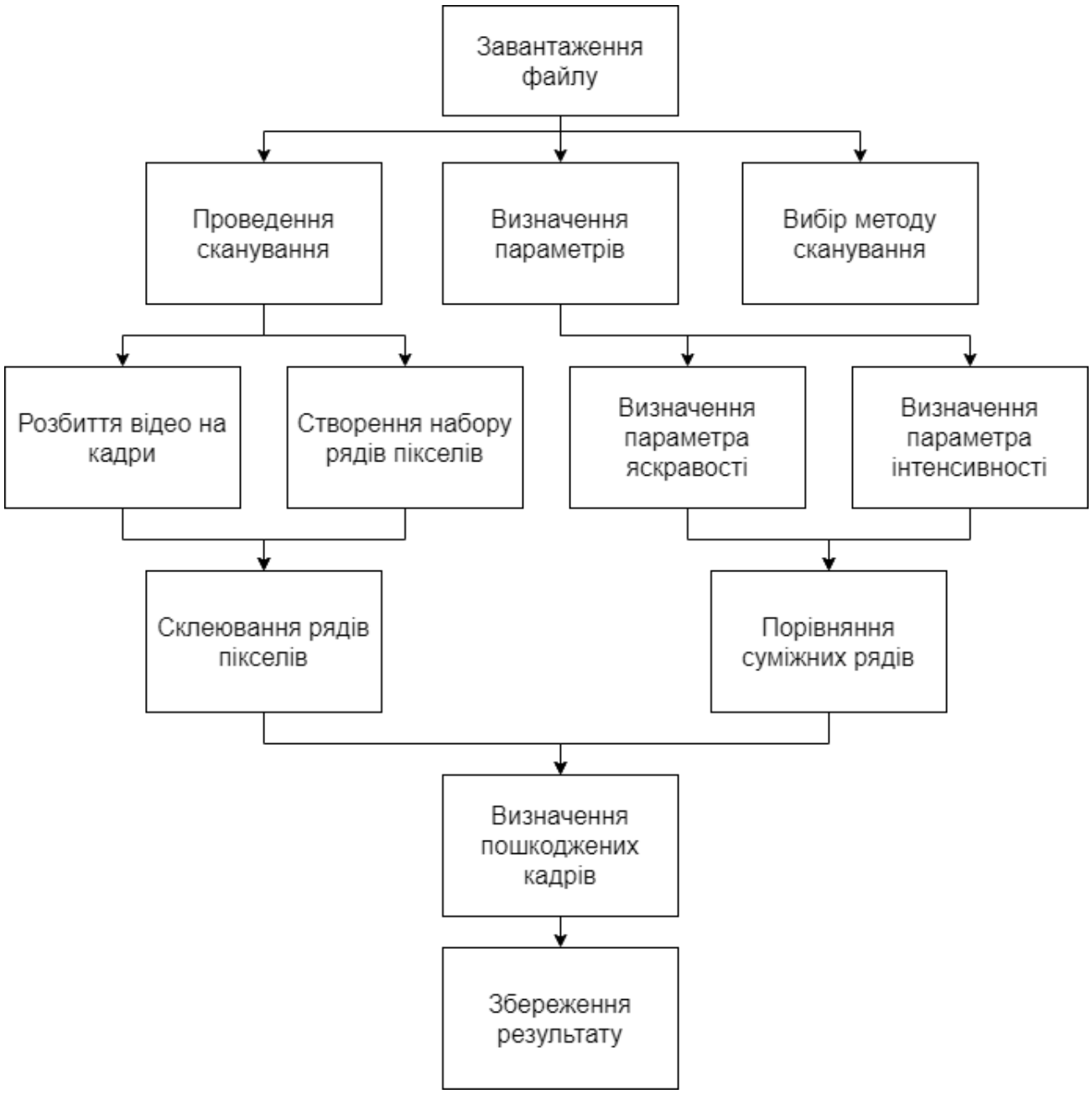
ДОДАТОК А

«Спосіб перевірки цілісності відео файлів»

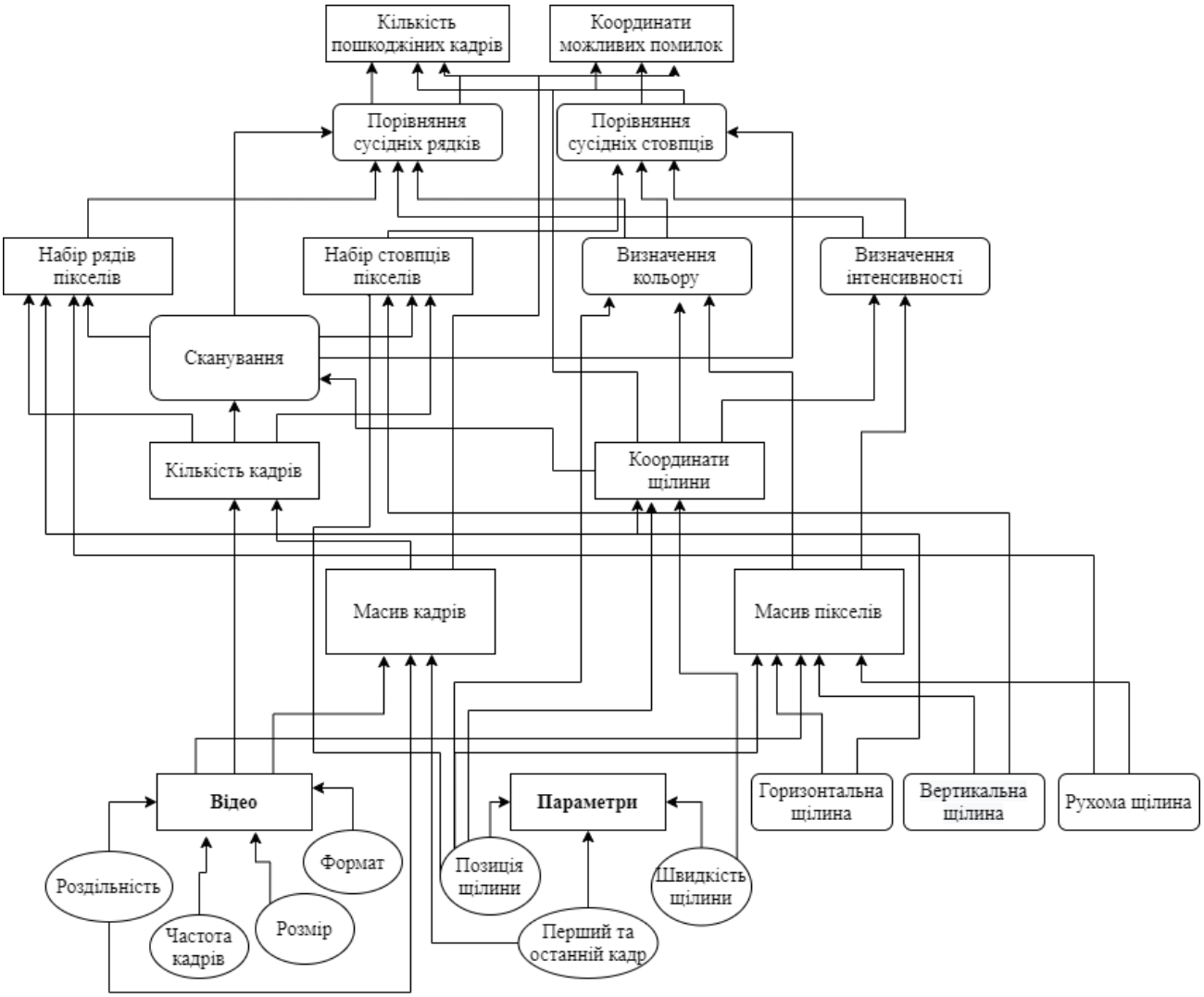
КОПІЇ ГРАФІЧНИХ МАТЕРІАЛІВ

Аркушів 3

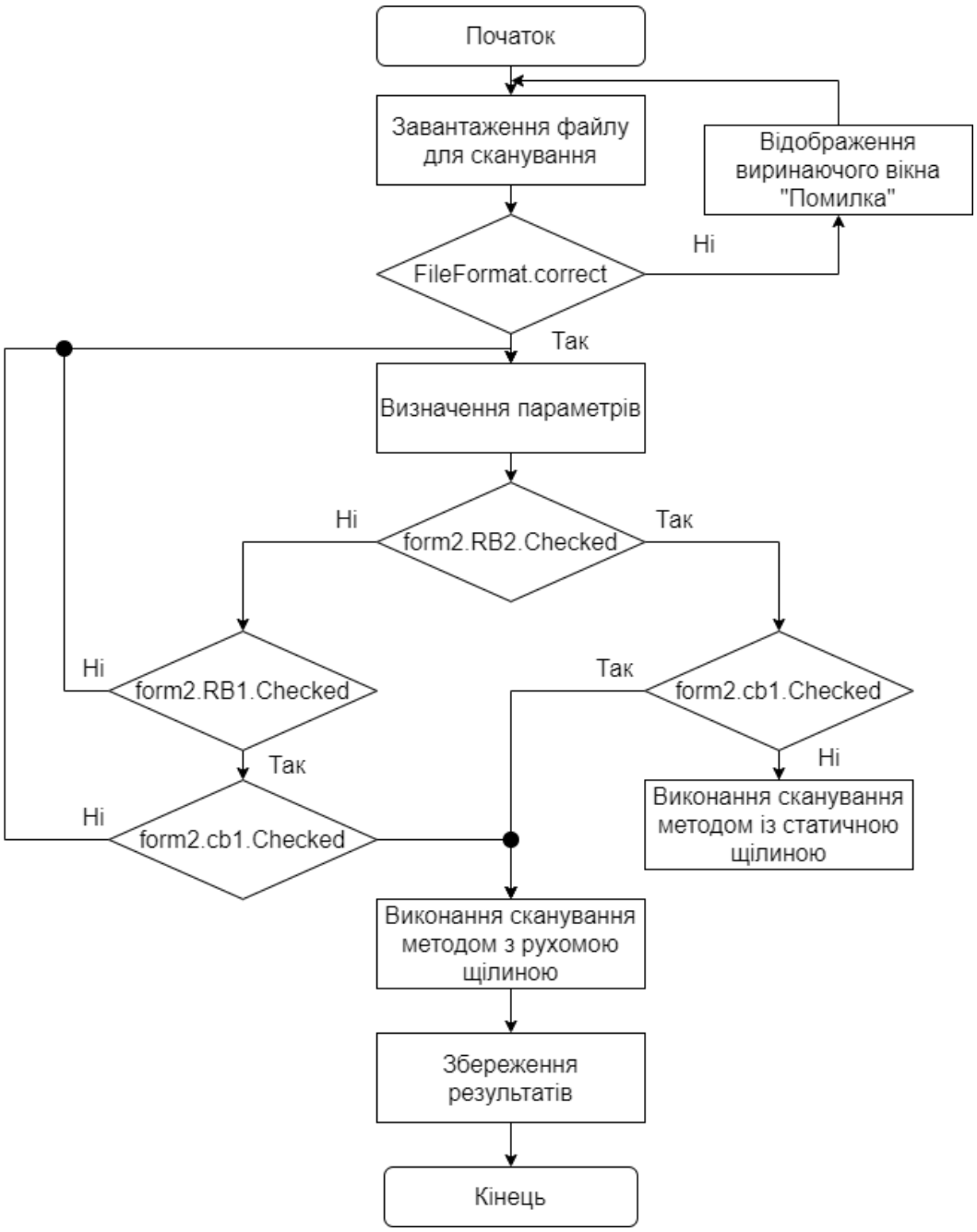
Київ – 2020



					ІАЛЦ.467200.004 Д1					
Зм.	Аркуш	№ Докум.	Підпис	Дата						
Розробив		Кішка М.І.			Спосіб перевірки цілісності відео файлів. Структурна схема програми					
Перевірив		Регіда П. Г.								
Т. Контр.						Аркуш 1		Аркушів 1		
						КПІ ім. Сікорського ФІОТ Група ІО-64				
Н. контр.		Сімоненко В.П.								
Затвердив		Регіда П. Г.								



					ІАЛЦ.467200.005 Д2				
Зм.	Аркуш	№ Докум.	Підпис	Дата	Спосіб перевірки цілісності відео файлів. Блок-схема процедур та параметрів				
Розробив		Кішка М.І.							
Перевірив		Регіда П. Г.							
Т. Контр.						Аркуш 1		Аркушів 1	
						КПІ ім. Сікорського ФІОТ Група ІО-64			
Н. контр.		Сімоненко В.П.							
Затвердив		Регіда П. Г.							



					ІАЛЦ.467200.006 ДЗ				
Зм.	Аркуш	№ Докум.	Підпис	Дата	Спосіб перевірки цілісності відео файлів. Блок-схема процесу сканування відео файлу				
Розробив		Кішка М.І.							
Перевірив		Регіда П. Г.							
Т. Контр.						Аркуш 1		Аркушів 1	
						КПІ ім. Сікорського ФІОТ Група ІО-64			
Н. контр.		Сімоненко В.П.							
Затвердив		Регіда П. Г.							

ДОДАТОК Б

«Спосіб перевірки цілісності відео файлів»

ЛІСТИНГ ПРОГРАМИ

Аркушів 12

Київ - 2020

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, MPlayer, ExtCtrls, StdCtrls, Buttons;
  type
  TPanel = class (ExtCtrls.TPanel)
    public
      property Canvas; // взлом канваса
  end;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    Exit1: TMenuItem;
    Exit2: TMenuItem;
    Open1: TMenuItem;
    Save1: TMenuItem;
    About1: TMenuItem;
    About2: TMenuItem;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    MediaPlayer1: TMediaPlayer;
    Edit1: TEdit;
    Button1: TButton;
    ScrollBox1: TScrollBox;
    Image2: TImage;
    Options1: TMenuItem;
    Parametrs1: TMenuItem;
    Edit2: TEdit;
    Button2: TButton;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    Button4: TButton;
    CheckBox1: TCheckBox;
    Panel1: TPanel;
    Button5: TButton;
    Edit5: TEdit;
```

					ІАЛЦ.467200.007 Д4		
Зм.	Аркуш	№ Докум.	Підпис	Дата	Спосіб перевірки цілісності відео файлів Додаток Б.		
Розробив		Кішка М. І.				Аркуш 1	Аркушів 12
Перевірив		Регіда П. Г.					
Т. Контр.						КПІ ім. Сікорського ФІОТ Група ІО-64	
Н. контр.		Сімоненко В.П.					
Затвердив		Регіда П. Г.					

```
    StaticText4: TStaticText;
    Bevel1: TBevel;
    Button3: TButton;
    procedure Exit2Click(Sender: TObject);
    procedure Open1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure Parametrs1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  j,i:integer;
  SlitPos: real;
  ADC : HDC;
  GetPoint:TPoint;
  BMP,bmp2 : TBitmap;
  H,W,L:integer;
  V : real;
  F_s, F_l : integer;
  flag:boolean;


implementation

uses Unit2;

{$R *.dfm}

procedure TForm1.Exit2Click(Sender: TObject);
begin
Form1.Close
end;
```

```
procedure PanelToImage;
begin

  form1.MediaPlayer1.TimeFormat := tfFrames;
  form1.MediaPlayer1.Display := form1.Panel1;
   with bmp.Canvas do
    CopyRect(ClipRect,form1.Panel1.Canvas,ClipRect);

  Application.ProcessMessages;

end;


procedure VerticalVideo;

var slitpos_old:real;

begin
 V:= strtofloat (form2.Edit2.Text);
 F_s:= strtoint (form2.Edit3.Text);
 F_l:= strtoint (form2.Edit4.Text);


 SlitPos:=100;
  SlitPos_old:=100;

 repeat
  BMP := TBitmap.Create;

    if form2.RadioButton2.Checked then
  begin
  BMP.Width:= L; // this was the size of my video file
  form1.Image2.Width:=L;
  end;


  if form2.RadioButton1.Checked then
  begin
  BMP.Width := W; // it is also the size of Panel1
  form1.Image2.Width:=W;
  end;



    BMP.Height := H; // it is also the size of Panel1
```

```
 BMP2 := TBitmap.Create;
 bmp2.Height:=H;
 bmp2.Width:=F_l-F_s;

form1.Edit1.Text:= floattostr (slitpos_old);
 for i:=F_s to F_l do
 begin
  form1.MediaPlayer1.Position:=i;
  PanelToImage;
  form1.Edit2.Text:= inttostr (i);
  if form2.checkbox1.Checked=true then
   begin
    SlitPos:=SlitPos+V;
 if SlitPos>W-1 then v:=-v;
 if (v<0) and (SlitPos<1) then v:=-v;
   end;

   begin
     for j:=1 to H do
     bmp2.canvas.Pixels[i-f_s,j]:=bmp.Canvas.Pixels[round(slitpos),j];
   end;
for j:=1 to H do
 form1.Image2.Canvas.Pixels[i-f_s,j]:=bmp.Canvas.Pixels[round(slitpos),j];
 end;

 bmp.Free;
 bmp2.SaveToFile(intToStr(round(slitpos_old))+'.bmp');
 bmp2.Free;

 slitpos_old:=slitpos_old+1;
  slitpos:=slitpos_old;
until flag;

end;

procedure HorizontalVideo;
var slitpos_old:real;
k:integer;

begin
 V:= strtofloat (form2.Edit2.Text);
 F_s:= strtoint (form2.Edit3.Text);
```

```
F_l:= strtoint (form2.Edit4.Text);
SlitPos:=1;
 SlitPos_old:=1;
 v:=1;
 k:=0;

repeat
form1.Edit1.Text:= inttostr (k);
 BMP := TBitmap.Create;
 BMP2 := TBitmap.Create;
 bmp2.Height:=H;
 if form2.RadioButton2.Checked then
 begin
 BMP.Width:= L; // this was the size of my video file
 form1.Image2.Width:=L;
 end;

 if form2.RadioButton1.Checked then
 begin
 BMP.Width := W; // it is also the size of Panel1
 form1.Image2.Width:=W;
 end;

 slitpos:=1;
for i:=F_s+k to F_s+H+k do

 begin
  form1.MediaPlayer1.Position:=i;
  PanelToImage;
  form1.Edit2.Text:= inttostr (i);
      SlitPos:=SlitPos+V;

    begin
     for j:=1 to W do
     bmp2.Canvas.Pixels[j,i-f_s-k]:=bmp.Canvas.Pixels[j,round(slitpos)];
    end;

for j:=1 to W do
 form1.Image2.Canvas.Pixels[j,i-f_s]:=bmp.Canvas.Pixels[j,round(slitpos)];
 end;

 bmp.Free;
 bmp2.SaveToFile(intToStr(round(k))+'.bmp');
 bmp2.Free;
```

```
    k:=k+1;
 until flag;
end;

procedure SphericVideo;
var d:real;
k:integer;

begin
 V:= strtofloat (form2.Edit2.Text);
 F_s:= strtoint (form2.Edit3.Text);
 F_l:= strtoint (form2.Edit4.Text);

k:=1;

 repeat

 begin
 form2.RadioButton1.Checked:=true;
V:=strtofloat(form1.edit5.text);
d:=(sqrt(w*w+h*h)/2);
 F_s:= strtoint (form2.Edit3.Text)+k;
 F_l:= strtoint (form2.Edit4.Text);
  BMP := TBitmap.Create;

  if form2.RadioButton2.Checked=true then
  begin
  BMP.Width:= L; // this was the size of my video file
  form1.Image2.Width:=L;
  end;


  if form2.RadioButton1.Checked=true then
  begin
  BMP.Width := W; // it is also the size of Panel1
  form1.Image2.Width:=W;
  end;

  BMP.Height := H; // it is also the size of Panel1

 SlitPos:=0;
i:=F_s;
 repeat
  form1.Edit1.Text:= inttostr (f_s);
begin
```

```
 form1.MediaPlayer1.Position:=i;
 PanelToImage;
 form1.Edit2.Text:= inttostr (i);
 SlitPos:=SlitPos+V;
begin
  for j:=0 to H do
  if j<slitpos then
  begin
  form1.Image2.Canvas.Pixels[round(sqrt(sqr(slitpos)-j*j))+w div 2,j+h div
2]:=bmp.Canvas.Pixels[round(sqrt(sqr(slitpos)-j*j))+w div 2,j+h div 2];
  form1.Image2.Canvas.Pixels[-round(sqrt(sqr(slitpos)-j*j))+w div 2,j+ h div
2]:=bmp.Canvas.Pixels[-round(sqrt(sqr(slitpos)-j*j))+w div 2,j+h div 2];
  form1.Image2.Canvas.Pixels[round(sqrt(sqr(slitpos)-j*j))+w div 2,-j+h div
2]:=bmp.Canvas.Pixels[round(sqrt(sqr(slitpos)-j*j))+w div 2,-j+h div 2];
  form1.Image2.Canvas.Pixels[-round(sqrt(sqr(slitpos)-j*j))+w div 2,-j+ h div
2]:=bmp.Canvas.Pixels[-round(sqrt(sqr(slitpos)-j*j))+w div 2,-j+h div 2];
  end;

  for j:=0 to H do
  if j<slitpos then
  begin
  form1.Image2.Canvas.Pixels[j+w div 2,round(sqrt(sqr(slitpos)-j*j))+h div
2]:=bmp.Canvas.Pixels[j+w div 2,round(sqrt(sqr(slitpos)-j*j))+h div 2];
  form1.Image2.Canvas.Pixels[j+w div 2,-round(sqrt(sqr(slitpos)-j*j))+h div
2]:=bmp.Canvas.Pixels[j+w div 2,-round(sqrt(sqr(slitpos)-j*j))+h div 2];
  form1.Image2.Canvas.Pixels[-j+w div 2,round(sqrt(sqr(slitpos)-j*j))+h div
2]:=bmp.Canvas.Pixels[-j+w div 2,round(sqrt(sqr(slitpos)-j*j))+h div 2];
  form1.Image2.Canvas.Pixels[-j+w div 2,-round(sqrt(sqr(slitpos)-j*j))+h div
2]:=bmp.Canvas.Pixels[-j+w div 2,-round(sqrt(sqr(slitpos)-j*j))+h div 2];

  end;

end;
end;
i:=i+1;

until slitpos>d;
end;

  bmp.Free;
 // bmp2.SaveToFile(intToStr(round(k))+'.bmp');
 // bmp2.Free;
 form1.Image2.Picture.SaveToFile(intToStr(round(f_s))+'.bmp');
    k:=k+1;
 until flag;
```

```
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
flag:=true;
end;

procedure TForm1.Open1Click(Sender: TObject);
begin

 if form1.OpenDialog1.Execute=true  then
 begin

 form1.MediaPlayer1.FileName:=form1.OpenDialog1.FileName;
 form1.MediaPlayer1.Open;
 form1.MediaPlayer1.TimeFormat:=tfFrames;

 MediaPlayer1.DisplayRect := form1.panel1.ClientRect;

H:=MediaPlayer1.DisplayRect.Bottom-MediaPlayer1.DisplayRect.Top;
W:=MediaPlayer1.DisplayRect.Right-MediaPlayer1.DisplayRect.Left;
L:=form1.MediaPlayer1.Length;

form1.Image2.Width:=L;
form1.Image2.Height:=H;
{form1.ScrollBox1.Height:=H;}
if form2.Edit1.Text='' then
form2.Edit1.Text:=inttostr (W div 2);
if form2.Edit4.Text='' then
form2.Edit4.Text:=inttostr (L);

 end;

end;

procedure TForm1.Button1Click(Sender: TObject);

begin
 V:= strtofloat (form2.Edit2.Text);
 F_s:= strtoint (form2.Edit3.Text);
 F_l:= strtoint (form2.Edit4.Text);
  BMP := TBitmap.Create;

  if form2.RadioButton2.Checked then
  begin
```

```
 BMP.Width:= L; // this was the size of my video file
 form1.Image2.Width:=L;
 end;

 if form2.RadioButton1.Checked then
 begin
 BMP.Width := W; // it is also the size of Panel1
 form1.Image2.Width:=W;
 end;

 BMP.Height := H; // this was the size of my video file

SlitPos:=strtoint (form2.Edit1.Text);

 for i:=F_s to F_l do
 begin
 MediaPlayer1.Position:=i;
 PanelToImage;
 form1.Edit2.Text:= inttostr (i);


 if form2.checkbox1.Checked=true then
 SlitPos:=SlitPos+V;


 if form2.radiobutton2.Checked=true then  begin   //вертик
  for j:=1 to H do
  form1.Image2.Canvas.Pixels[i-f_s,j]:=bmp.Canvas.Pixels[round(slitpos),j];
  if SlitPos>W-1 then v:=-v;
  if (v<0) and (SlitPos<1) then v:=-v;
  end;
 if  form2.radiobutton1.Checked=true then    //гориз
  for j:=1 to W do
  form1.Image2.Canvas.Pixels[j,i-f_s]:=bmp.Canvas.Pixels[j,round(slitpos)];

  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Image2.Parent.DoubleBuffered := true;
end;

procedure TForm1.Save1Click(Sender: TObject);
begin
```

```
 if form1.SaveDialog1.Execute=true  then
 form1.Image2.Picture.SaveToFile(Savedialog1.FileName+'.bmp');
 end;

procedure TForm1.Parametrs1Click(Sender: TObject);
begin
form2.show;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
if checkbox1.Checked=true then form1.image2.Visible:=false;
if checkbox1.Checked=false then form1.image2.Visible:=true;

end;

procedure TForm1.Button4Click(Sender: TObject);
begin
if form2.RadioButton1.Checked then horizontalvideo;
if form2.RadioButton2.Checked then verticalvideo;

end;

procedure TForm1.Button5Click(Sender: TObject);
var
d:real;
begin
V:=strtofloat(edit5.text);
d:=(sqrt(w*w+h*h)/2);
 F_s:= strtoint (form2.Edit3.Text);
 F_l:= strtoint (form2.Edit4.Text);
  BMP := TBitmap.Create;

  if form2.RadioButton2.Checked=true then
  begin
  BMP.Width:= L; // this was the size of my video file
  form1.Image2.Width:=L;
  end;

  if form2.RadioButton1.Checked=true then
  begin
  BMP.Width := W; // it is also the size of Panel1
  form1.Image2.Width:=W;
  end;
  BMP.Height := H; // it is also the size of Panel1
```

```
 SlitPos:=0;

i:=F_s;
 repeat
begin
 MediaPlayer1.Position:=i;
 PanelToImage;
 form1.Edit2.Text:= inttostr (i);
 SlitPos:=SlitPos+V;

begin
  for j:=0 to H do
  if j<slitpos then
  begin
  form1.Image2.Canvas.Pixels[round(sqrt(sqr(slitpos)-j*j))+w div 2,j+h div
2]:=bmp.Canvas.Pixels[round(sqrt(sqr(slitpos)-j*j))+w div 2,j+h div 2];
  form1.Image2.Canvas.Pixels[-round(sqrt(sqr(slitpos)-j*j))+w div 2,j+ h div
2]:=bmp.Canvas.Pixels[-round(sqrt(sqr(slitpos)-j*j))+w div 2,j+h div 2];
  form1.Image2.Canvas.Pixels[round(sqrt(sqr(slitpos)-j*j))+w div 2,-j+h div
2]:=bmp.Canvas.Pixels[round(sqrt(sqr(slitpos)-j*j))+w div 2,-j+h div 2];
  form1.Image2.Canvas.Pixels[-round(sqrt(sqr(slitpos)-j*j))+w div 2,-j+ h div
2]:=bmp.Canvas.Pixels[-round(sqrt(sqr(slitpos)-j*j))+w div 2,-j+h div 2];

  end;

  for j:=0 to H do
  if j<slitpos then
  begin
  form1.Image2.Canvas.Pixels[j+w div 2,round(sqrt(sqr(slitpos)-j*j))+h div
2]:=bmp.Canvas.Pixels[j+w div 2,round(sqrt(sqr(slitpos)-j*j))+h div 2];
  form1.Image2.Canvas.Pixels[j+w div 2,-round(sqrt(sqr(slitpos)-j*j))+h div
2]:=bmp.Canvas.Pixels[j+w div 2,-round(sqrt(sqr(slitpos)-j*j))+h div 2];
  form1.Image2.Canvas.Pixels[-j+w div 2,round(sqrt(sqr(slitpos)-j*j))+h div
2]:=bmp.Canvas.Pixels[-j+w div 2,round(sqrt(sqr(slitpos)-j*j))+h div 2];
  form1.Image2.Canvas.Pixels[-j+w div 2,-round(sqrt(sqr(slitpos)-j*j))+h div
2]:=bmp.Canvas.Pixels[-j+w div 2,-round(sqrt(sqr(slitpos)-j*j))+h div 2];

  end;
end;
end;
i:=i+1;

until slitpos>d;

  end;
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
sphericvideo;
end;

end.


unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Buttons;

type
  TForm2 = class(TForm)
    Edit1: TEdit;
    StaticText1: TStaticText;
    Button1: TButton;
    Edit2: TEdit;
    StaticText2: TStaticText;
    Edit3: TEdit;
    StaticText3: TStaticText;
    StaticText4: TStaticText;
    Edit4: TEdit;
    CheckBox1: TCheckBox;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form2: TForm2;
implementation
uses Unit1;
{$R *.dfm}
procedure TForm2.Button1Click(Sender: TObject);
begin
form2.Close;
end;
end.
```